IBM WebSphere Business Integration Adapters

Adapter for Healthcare Data Protocols User Guide

Version 1.1.0

IBM WebSphere Business Integration Adapters

Adapter for Healthcare Data Protocols User Guide

Version 1.1.0

Note!
Before using this information and the product it supports, read the information in Appendix E, "Notices," on page 131.

19December2003

This edition of this document applies to IBM WebSphere Integration Adapter for Healthcare Protocols, version 1.1.0, to all subsequent releases and modifications until otherwise indicated in new editions

To send us your comments about this document, email doc-comments@us.ibm.com. We look forward to hearing from you.

When you send information to IBM, you grant IBM a nonexclusive right to use or distribute the information in any way it believes appropriate without incurring any obligation to you.

Contents

About this document

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

This document describes the installation, configuration, business object development, and troubleshooting for the IBM WebSphere Business Integration Adapter for Healthcare Data Protocols.

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 Healthcare data protocols

Related documents

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

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

You can install documentation from the following sites:

- "For general adapter information; for using adapters with WebSphere message brokers (WebSphere MQ Integrator, WebSphere MQ Integrator Broker, WebSphere Business Integration Message Broker); and for using adapters with WebSphere Application Server:

  http://www.ibm.com/websphere/integration/wbiadapters/infocenter
- For using adapters with InterChange Server:

  http://www.ibm.com/websphere/integration/wicserver/infocenter

  http://www.ibm.com/websphere/integration/wbicollaborations/infocenter
- For more information about message brokers (WebSphere MQ Integrator Broker, WebSphere MQ Integrator, and WebSphere Business Integration Message Broker):

http://www.ibm.com/software/integration/mqfamily/library/manualsa/.

- For more information about WebSphere Application Server:
  http://www.ibm.com/software/webservers/appserv/library.html

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

Typographic conventions

This document uses the following conventions:

`courier font`	Indicates a literal value, such as a command name, filename, information that you type, or information that the system prints on the screen.
bold	Indicates a new term the first time that it appears.
italic, *`italic`*	Indicates a variable name or a cross-reference.
blue text	Blue text, which is visible only when you view the manual online, indicates a cross-reference hyperlink. Click any blue text to jump to the object of the reference.
ProductDir	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, `option[,...]` means that you can enter multiple, comma-separated options.
`< >`	In a naming convention, angle brackets surround individual elements of a name to distinguish them from each other, as in `<server_name><`*`connector_name`*`>tmp.log`.
`/, \`	In this document, backslashes (\) are used as the convention for directory paths. For UNIX installations, substitute slashes (/) for backslashes. All product pathnames are relative to the directory where the product is installed on your system.
`%text%` and *`$text`*	Text within percent (%) signs indicates the value of the Windows `text` system variable or user variable. The equivalent notation in a UNIX environment is *`$text`*, indicating the value of the *`text`* UNIX environment variable.

New in this release

New in release 1.1

Updated December 2003. This adapter includes:

- Support for HP-UX
- Business object wrappers to use when mapping repeating HL7 data elements, instead of mapping each repeating data element to a primitive business object attribute.
- Support for extended length segment length ID in the HL7 data handler.
- A business object extractor utility. This is a batch file that is run against the master file of industry-specific business objects. The utility enables you to extract a subset of industry-specific business objects, offering an alternative method for constructing your application-specific business objects.
- Adapter installation information has been removed from this guide. See Chapter 2 for the new location of this information.

New in release 1.0

This is the first release of the product.

Chapter 1. Overview

The connector for the Healthcare datahandlers, Healthcare Level Seven (HL7) and National Council for Drug Control programs (NCPDP) message standard, is a runtime component of the WebSphere Business Integration Adapter for Healthcare Data Protocols. Both HL7 and NCPDP message standards refer to other well-defined standards, such as various ISO and ANSI standards. The connector allows the WebSphere integration broker to exchange business objects with healthcare-enabled business processes.

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 the healthcare data protocol operates with both the WebSphere InterChange Server (ICS) and WebSphere MQ Integrator Broker integration brokers.

The connector for the healthcare data protocol allows the IBM InterChange Server or IBM WebSphere MQ Integrator Broker to exchange business objects with applications that send or receive data in the form of HL7/NCPDP messages.

Connector architecture

The connector allows WebSphere business processes to asynchronously exchange business objects with applications that issue or receive healthcare (HL7/NCPDP) data protocol messages when changes to data occur.

As shown in Figure 1, the connector interacts with several components whose collective purpose is to bridge the world of WebSphere business objects with that of healthcare messages.

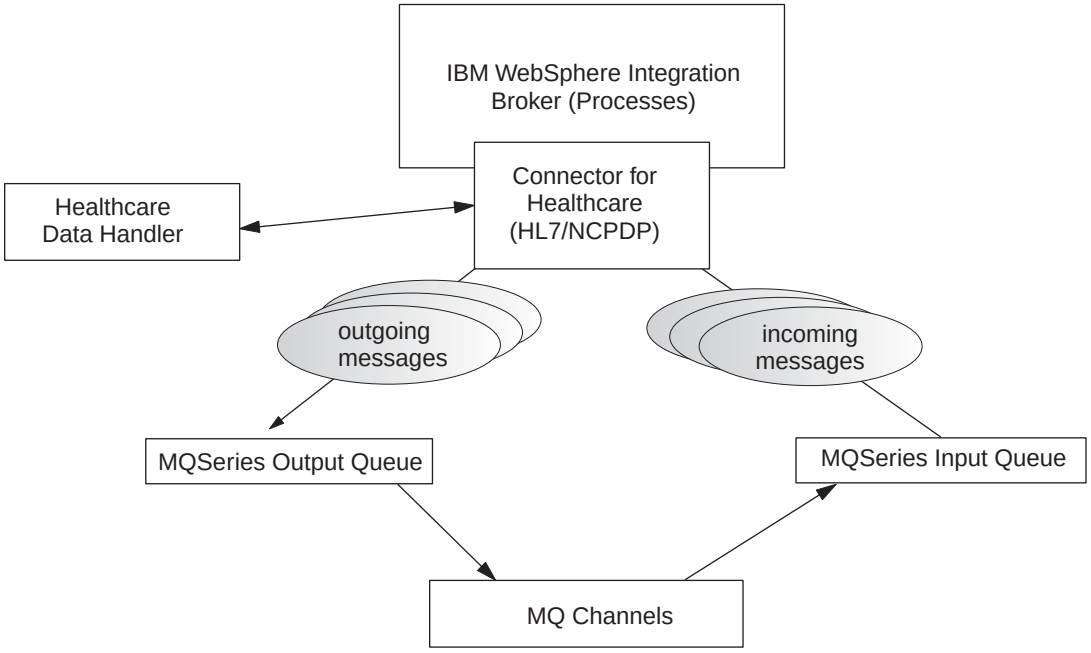

Figure 1. Healthcare data handler architecture

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

Connector for Healthcare

The connector for healthcare 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 healthcare 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 healthcare 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 `gotApplEvents()` method.

Healthcare data handler

The connector calls the heathcare data handler to convert business objects into HL7/NCPDP messages and vice versa.

WebSphere MQ

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

MQ channels

The WebSphere MQ event queues exchange messages with the WebSphere MQ Interface. The software integrates WebSphere MQ messaging capabilities with HL7/NCPDP message types, performing delivery, acknowledgement, queue management, timestamping, and other functions.

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 healthcare message from an integration broker.
2. The connector passes the business object to the data handler.
3. The data handler converts the healthcare business object into an HL7/NCPDP-compliant message.

4. The connector dispatches the HL7/NCPDP-compliant message to the WebSphere MQ output queue.
5. The JMS layer makes the appropriate calls to open a queue session and routes the message to the MQ Series input queue.

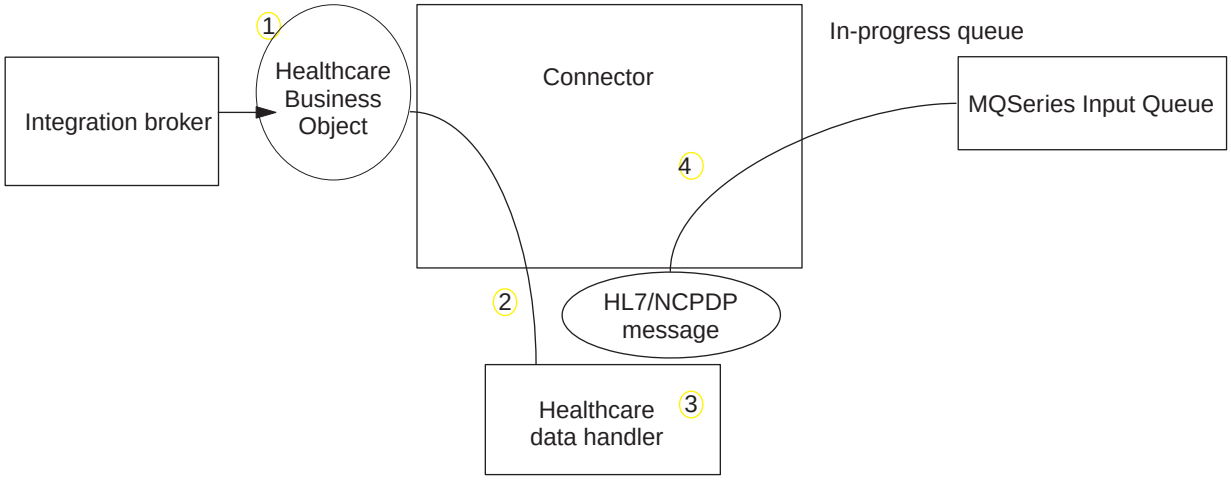

Figure 2. Application-connector communication method: Message request

Event delivery

Figure 2 illustrates the message return communication.

1. The polling method retrieves the next applicable message from the WebSphere MQ input queue.
2. The data handler converts the message into a business object.
3. The HL7 data handler receives the business object and sets the verb in it to the default verb specified in the data handler-specific meta-object.
4. 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.

Event handling

For event notification, the connector detects an event written to a queue by an application rather than by a database trigger. when messages are stored them in 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 healthcare 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 Healthcare is notified of its outcome.

If the connector for healthcare 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`	Message type	`DATAGRAM`

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

MQMD Field (message descriptor)	Description	Supported values (multiple values should be OR'd)
Report	Options for report message requested	You can specify one or both of the following: • MQRO_PAN The connector sends a report message if the business object can be successfully processed. • MQRO_NANThe connector sends a report message if an error occurred while processing the business object. You can specify one of the following to control how the correlation ID of the report message is to be set: • MQRO_COPY_MSG_ID_TO_CORREL_IDThe connector copies the message ID of the request message to the correlation ID of the report. This is the default action. • MQRO_PASS_CORREL_IDThe 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.
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	Message type	REPORT

Table 2. Structure of the report returned to the requesting application (continued)

MQMD field	Description	Supported values (multiple values should be OR'd)
`feedback`	Type of report	One of the following: • `MQRO_PAN` If the business object is successfully processed. • `MQRO_NANI`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 connector`DeliveryTransport` 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 gotApplEvent() method within its pollForEvents() method) and the time it updates the event store by deleting the event record (or perhaps updating it with an "event posted" status). If a failure occurs in this window, the event has been sent but its event record remains in the event store with an "in progress" status. When the connector restarts, it finds this event record still in the event store and sends it, resulting in the event being sent twice.

You can configure the guaranteed-event-delivery feature for a JMS-enabled connector with, or without, a JMS event store. To configure the connector for guaranteed event delivery, see "Enabling guaranteed event delivery" on page 23.

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 healthcare data handler, and depending on the requirements specified in the subscription meta-object, the connector can transform HL7 and NCPDP objects before issuing them.

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 `ReplyToQueue`.
Persistence	Message persistence	`MQPER_PERSISTENT`[a]
Expiry	Message lifetime	`MQEI_UNLIMITED`[a]

[a] Indicates constant defined by IBM.

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

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

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

Table 4. Response Message Descriptor Header (MQMD)

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

[a] Indicates constant defined by IBM.

Table 5. Population of response message

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

Table 6. Feedback codes and response values

WebSphere MQ feedback code	Equivalent WebSphere business integration 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 20.

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 FAIL_RETRIEVE_BY_CONTENT	(Optional) An error message.
	MULTIPLE_HITS SUCCESS	A serialized business object.

Error handling

All error messages generated by the connector are stored in a message file named BIA_HealthcareConnector.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 15, 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 `gotApplEvent()` or `executeCollaboration()`.
Level 3	Use this level for trace messages that provide information regarding message-to-business-object and business-object-to-message conversions or provide information about the delivery of the message to the output queue.
Level 4	Use this level for trace messages that identify when the connector enters or exits a function.
Level 5	Use this level for trace messages that indicate connector initialization, represent statements executed in the application, indicate whenever a message is taken off of or put onto a queue, or record business object dumps.

Chapter 2. Configuring the connector

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

Compatibility

The adapter framework that an adapter uses must be compatible with the version of the integration broker (or brokers) with which the adapter is communicating. Version 1.1 of the Adapter for Healthcare Data Protocols is supported on the following adapter framework and integration brokers:

- **Adapter framework**: WebSphere Business Integration Adapter Framework, versions 2.1, 2.2, 2.3.x, and 2.4.
- **Integration brokers:**
  - WebSphere InterChange Server, version 4.2, 4.21, and 4.22.
  - WebShpere MQ Integrator, version 2.1.0
  - WebSphere MQ Integrator broker, version2.1.0
  - WebSphere Business Integration Message Broker, version 2.1 and 5.0
  - WebSphere Application Server Enterprise, version 5.0.2, with WebSphere Studio Application Developer Integration Edition, version 5.0.1

See the Release Notes for any exceptions.

Note:

For instructions on installing the integration broker and its prerequisites, see the following documentation.

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

For message brokers (WebSphere MQ Integrator Broker, WebSphere MQ Integrator, and WebSphere Business Integration Message Broker), see Implementing Adapters with WebSphere Message Brokers, and the installation documentation for the message broker. Some of this can be found at the following Web site:

http://www.ibm.com/software/integration/mqfamily/library/manualsa/

For WebSphere Application Server, see Implementing Adapters with WebSphere Application Server and the documentation at:

http://www.ibm.com/software/webservers/appserv/library.html.

Prerequisites

Prerequisite software

The following software must be installed before installing and configuring the Adapter for Healthcare Data Protocols:

- For Windows systems: Windows 2000

  Note: Beginning with the 1.1 version, the Adapter for Healthcare Data Protocols is no longer supported on Microsoft Windows NT.
- For Unix systems: Solaris 7.0 or 8.0, or AIX 5.1 or 5.2, or HP-UX11i.

Installing the adapter and related files

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

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

Installed file structures

The following subsections describe the file stuctures used by the connector on a UNIX or Windows system.

Installed file structure on UNIX

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

Table 8. Installed UNIX file structure for the connector

Subdirectory of *ProductDir*	Description
`connectors/Healthcare/BIA_CW_Healthcare.jar`	Connector jar files
`connectors/Healthcare/start_Healthcare.sh`	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 `mqsiremotestopadapter` to stop the connector.
`connectors/messages/BIA_HeathcareConnector.txt`	Connector message file
`repository/Healthcare/dependencies/ BIA_HL7MTEventMap.cfg`	Configuration file
`repository/Healthcare/dependencies/ BIA_HL7I18N.cfg`	Configuration file

Table 8. Installed UNIX file structure for the connector (continued)

Subdirectory of *ProductDir*	Description
connectors/Heathcare/ BIA_CWHealthcareDataHandler.jar	The Healthcare data handler
/lib	Contains the WBIA. jar file.
/bin	Contains the CWConnEnv.sh file.

Installed file structure on Windows

Table 9 describes the Windows file structure used by the connector.

Table 9. Installed Windows file structure for the connector

Subdirectory of ProductDir	Description
connectors\Healthcare\BIA_CW_Healthcare.jar	Connector jar files
connectors\Healthcare\start_Healthcare.bat	The startup file for the connector.
connectors\messages\BIA_HeathcareConnector.txt	Connector message file
connectors\Healthcare\BIA_CW_HealthcareDataHandler.jar	The Healthcare data handler
repository\DataHandlers\MO_DataHandler_Default.txt	Data handler default object
repository\Healthcare/dependencies\BIA_HL7MTEventMap.cfg	Configuration file
repository\Healthcare\dependencies\BIA_HL7I18N.cfg	Configuration file
\lib	Contains the WBIA. jar file.
\bin	Contains the CWConnEnv.bat file.

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 99

Standard connector properties

Standard configuration properties provide information that all connectors use. See Appendix A, "Standard configuration properties for connectors," on page 83 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 heathcare. 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 19	*Login password*		No
"ApplicationUserID" on page 19	*Login user ID*		No
"ArchiveQueue" on page 19	*Queue to which copies of successfully processed messages are sent*	`queue://CrossWorlds. QueueManager/MQCONN.`	No
"BOPref" on page 19	*Business object preference*	HL7	Yes
"Channel" on page 19	*MQ server connector channel*		Yes
"ClassName" on page 20	*Data handler class name*	`com.ibm.adapters. datahandlers.hl7. HL7DataHandler`	No
"ConfigurationMetaObject" on page 20	*Name of configuration meta-object*		Yes
"Component delimiter" on page 20	*User defined*	^	
"DataHandlerConfigMO" on page 20	*Data handler meta-object*	`MO_DataHandler _Default`	Yes
"DataHandlerMimeType" on page 20	*MIME type of file*	`HL7`	No
"DefaultVerb" on page 20	*Any verb supported by the connector.*	`Create`	
"Dummy string" on page 20	*User defined*		
"ErrorQueue" on page 20	*Queue for unprocessed messages*	`queue://crossworlds. Queue.manager/ MQCONN.ERROR`	No
"Field delimiter" on page 21	Default field delimiter for all types.	The default value is `"|"`	
"HostName" on page 21	*WebSphere MQ server*	The default is `"|"`.	No
"InDoubtEvents" on page 21	`FailOnStartup Reprocess Ignore LogError`	`Reprocess`	No
"InputQueue" on page 21	*Poll queues*	`queue://CrossWorlds. QueueManager/MQCONN.IN`	Yes
"InProgressQueue" on page 22	*In-progress event queue*	`queue://CrossWorlds. QueueManager/ MQCONN.IN_PROGRESS`	No
"PollQuantity" on page 22	*Number of messages to retrieve from each queue specified in the `InputQueue` property*	1	No
"Port" on page 22	*Port established for the WebSphere MQ listener*		No
"ReplyToQueue" on page 22	*Queue to which response messages are delivered when the connector issues requests*	`queue://CrossWorlds. QueueManager/ MQCONN.REPLYTO`	No

Table 10. Connector-specific configuration properties (continued)

Name	Possible values	Default value	Required
"Representation" on page 22	*Header or message object parsing.*	Can be one of the following:"Simplified" or "Native" Simplified signifies the use of a parsing method which provides fine object represent only to message headers. Native signifies the use of parsing method, which provides fine object represent to the whole message.	
"Repetition delimiter" on page 22	*User defined*	The default value is "~"	
"Subcomponent delimiter" on page 23	*User defined*	The default value is "&"	
"UnsubscribedQueue" on page 23	*Queue to which unsubscribed messages are sent*	`queue://CrossWorlds.QueueManager/MQCONN.UNSUBSCRIBE`	No
"UseDefaults" on page 23	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`

BOPref
Business object preferences prefix. The default is HL7.

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.

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

Default = `com.ibm.adapters.DataHandlers.hl7.HL7DataHandler`

Component delimiter
The default component delimiter for all flights. The default is ″^″.

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

Default = none.

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 = `HL7`

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`

Dummy string
The dummy string.

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 5. 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.

Field delimiter
The default delimiter for all types. The default is "|" the pipe symbol.

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.

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

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

With pollQuanity set to 2, the connector retrieves at most 2 messages from each queue per call to pollForEvents. For the first cycle (1 of 2), the connector retrieves the first message from each of MyQueueA, MyQueueB, and MyQueueC. That completes the first round of polling. The connector starts a second round of polling (2 of 2)

and retrieves one message each from MyQueueA and MyQueueC—it skips MqQueueB because that queue is now empty. After polling all queues twice, the call to the method pollForEvents is complete. The sequence of message retrieval is:

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

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

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

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

I118N
Specifies the path name of the BIA_HL7I18N.cfg file
..\Healthcare\dependencies\hl7\BIA_HL7I18N.cfg

MTEventMap
This configuration contains a map of (Message X event type) -> Message structure. The file is located at ...\Healthcare\dependencies\hl7\HL7MTEventMap.cfg.

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.

Repetition delimiter
Default repetition delimiter for all types. The default is "~".

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

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

Representation
Representation determines how the message is parsed, either "Simplified" or "Native." Simplified parses so that it represents only the message header. Native parses so that the entire message is represented. The default is "Simplified."

Subcomponent delimiter
Default subcomponent delimiter. The default is :&."

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

Default =Default sub component delimiter for all types
`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 25.

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 25

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

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

Connector property	Value
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 99.

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 24.
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`

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

Connector property	Value
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 99.

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 gotApplEvent() in the pollForEvents() method), it must determine if the current business object (received from gotApplEvents()) 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 gotApplEvent() 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.

Meta-object attributes configuration

The connector for HL7 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 `HL7_MTADT_A03_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 15 describes the meta-object properties.

Table 15. Static meta-object properties

Property name	Description
`CollaborationName`	The collaboration name must be specified in the application-specific text of the attribute for the business object/verb combination. For example, if you expect to handle synchronous requests for the business object Customer with the Create verb, the static meta-data object must contain an attribute named `HL7_MT`*`nnn`*`_Verb`, where *nnn* is the HL7 message type, for example, `HL7_MTADT_A03_Create`. The `HL7_MTADT_A03_Create` attribute must contain application-specific text that includes a name-value pair. For example, CollaborationName=MyCustomerProcessingCollab. See the "Application-specific information" on page 29 section for syntax details. Failure to do this results in runtime errors when the connector attempts to synchronously process a request involving the Customer business object. **Note:** This property is available only for synchronous requests.
`DoNotReportBusObj`	Optionally, you can include the `DoNotReportBusObj` property. By setting this property to `true`, all PAN report messages issued have a blank message body. This is recommended when you want to confirm that a request has been successfully processed but does not need notification of changes to the business object. This does not affect NAN reports. If this property is not found in the static meta-object, the connector defaults to `false` and populates the message report with the business object. **Note:** This property is available only for synchronous requests.
`InputFormat`	The input format is the message format to associate with the given business object. When a message is retrieved and is in this format, it is converted to the given business object if possible. If this format is not specified for a business object, the connector does not handle subscription deliveries for the given business object.
`OutputFormat`	The output format is set on messages created from the given business object. If a value for the `OutputFormat` property is not specified, the input format is used, if available. An `OutputFormat` property value defined in a dynamic child meta-object overrides the value defined in the static meta-object.
`InputQueue`	The input queue that the connector polls to detect new messages. You can use connector-specific properties to configure multiple InputQueues and optionally map different data handlers to each queue.
`OutputQueue`	The output queue is the queue to which messages derived from the given business object are delivered. An OutputQueue property value defined in a dynamic child meta-object overrides the value defined in the static meta-object.

Table 15. 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.

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

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 21) 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 = HL7_MTADT_A03_Create
Type = String
Cardinality = 1
MaxLength = 1
IsKey = false
IsForeignKey = false
IsRequired = false
AppSpecificInfo = InputQueue=//queue.manager/CompReceipts;
```

```
  DataHandlerClassName=com.crossworlds.
DataHandlers.HL7.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.

Static configuration meta object

A static configuration meta-object is specific to MQ connectivity. It provides the following services:

- Defines the output queue
- Defines the verb upon the retrevial of a message from the input queue for each BO. This BO_Verb association with an input message is specified on the attribute name.
- Specifies the format of data from both input queue and output queue fo reach BO. This information is specified in the ASI Property.

Table 16. Where IK=Is key, FK=Foreign key, IR=Is required, C=Cardinality, ML=Maximum length, and Def=Default

Attribute Name	Type	IK	FK	IR	C	ML	Def	ASI	Comment
Default	String	Yes	No	Yes	1	1		`OutputQueue =queue: //crossworlds .queue. manager/ MQCONN.OUT`	Output queue defnition
HL7 _Message _Create	String	No	No	No	1	7	Create	`InputFormat =MQSTR; OutputFormat =MQSTR`	
Object Event ID		No		No	1				Reserved for system use

A sample static meta-object

The static meta-object shown below configures the connector to convert HL7_MTADT_A03 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 HL7_MTADT_A03_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 HL7_MTADT_A03_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 = 3.0.0
Repositories = 1cHyILNuPTc=
[End]
[BusinessObjectDefinition]
Name = Sample_MO
Version = 3.0.0

[Attribute]
Name = Default
Type = String
Cardinality = 1
MaxLength = 1
IsKey = true
IsForeignKey = false
IsRequired = false
AppSpecificInfo = OutputQueue=queue://crossworlds.queue.manager/MQCONN.OUT;
   ResponseTimeout=5000;TimeoutFatal=true
IsRequiredServerBound = false
[End]
[Attribute]
Name = HL7_MTADT_A03_Create
Type = String
Cardinality = 1
MaxLength = 1
IsKey = false
IsForeignKey = false
IsRequired = false
AppSpecificInfo = InputFormat=NEW
IsRequiredServerBound = false
[End]
[Attribute]
Name = HL7_MTADT_A03_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 `HL7_MTADT_A03_Create`. Note that the application-specific text consists of semicolon-delimited name-value pairs

Table 17. Static meta-object structure for HL7_MTADT_A03_Create

Attribute name	Application-specific text
HL7_MTADT_A03_Create	`InputFormat=ORDER_IN; OutputFormat=ORDER_OUT; OutputQueue=QueueA; ResponseTimeout=10000; TimeoutFatal=False`

Table 18 shows a sample dynamic child meta-object for business object `HL7_MT_Create`.

Table 18. Dynamic child meta-object Structure for HL7_MTADT_A03_Create

Property name	Value
`OutputFormat`	`ORDER_OUT`

Table 18. Dynamic child meta-object Structure for HL7_MTADT_A03_Create (continued)

Property name	Value
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.

MO_DataHandler_Healthcare

The following meta-object is used to configure all heathcare data handlers.

Table 20. Where IK=Is key, FK=Foreign key, IR=Is required, C=Cardinality, ML=Maximum length, and Def=Default

Attribute Name	Type	IK	FK	IR	C	ML	Def	Comment
HL7	`MO_ DataHandler _HL7`	No		No	1	N/A		Attribute name designates a name for the MIME type of HL7 messages
Dummy Key	String	Yes	No	Yes	1	1		

MO_DataHandler_HL7

The following meta-object contains configuration properties for the HL7 data handler. BO_Prefix, Default Verb, and ClassName are used in the invocation of the data handler by the connector.

Table 21. Where IK=Is key, FK=Foreign key, IR=Is required, C=Cardinality, ML=Maximum length, and Def=Default

Attribute Name	Type	IK	FK	IR	C	ML	Def	Comment
BOPrefix	String	Yes	No	Yes	1	4	HL7	BO's prefix
Default Verb	String	No	No	No	1	7	Create	Verb set in BO
Class Name	String	No	No	No	1	255	`com.ibm .adapters .datahandlers .hl7 .HL7DataHandle`	Java class name of the HL7 data handler

Table 21. Where IK=Is key, FK=Foreign key, IR=Is required, C=Cardinality, ML=Maximum length, and Def=Default (continued)

Attribute Name	Type	IK	FK	IR	C	ML	Def	Comment
Representation	String	No	No	No	1	1	Simplified	Can be on of the following: ″Simplified″ or ″Native″ Simplified signifies the use of a parsing method which provides find object representation to only the message header. Native signifies the user of a parsing method which provides fine object representation to the whole message
Field delimiter	String	No	No	No	1	1	|	Default field delimiter for all types
Repetition Delimiter	String	No	No	No	1	1	~	Default repetition delimiter for all types
Component Delimiter	String	No	No	No	1	1	^	Default component delimiter for all types
Sub-component Delimiter	String	No	No	No	1	1	&	Default subcomponent delimiter for all types
MTEventMap	String	No	No	No	1	255	`file=...\ Healthcare\ dependencies\ HL7\ BIA_ HL7MTEventMap.cfg`	This configuration contains a map of (Message type X Event Type)->Message Structure

Table 21. Where IK=Is key, FK=Foreign key, IR=Is required, C=Cardinality, ML=Maximum length, and Def=Default (continued)

Attribute Name	Type	IK	FK	IR	C	ML	Def	Comment
I118N	String	No	No	No	1	255	`file=...\ Healthcare\ dependencies\ HL7\ BIA_HLI118N.cfg`	Specifies the path name of the `BIA_ HL7I118N.cfg` file. This files contains map information on ISO character set, escape secquence and Java name
DummyKey	String	Yes	No	No	1			

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 = HL7_MTADT_A03
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, HL7 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 HL7 header properties, are recognized in the dynamic meta-object.

Table 22. 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

Table 22. Dynamic meta-object header attributes (continued)

Header attribute name	Mode	Corresponding JMS header
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 23. Application-specific information for JMS property attributes

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

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

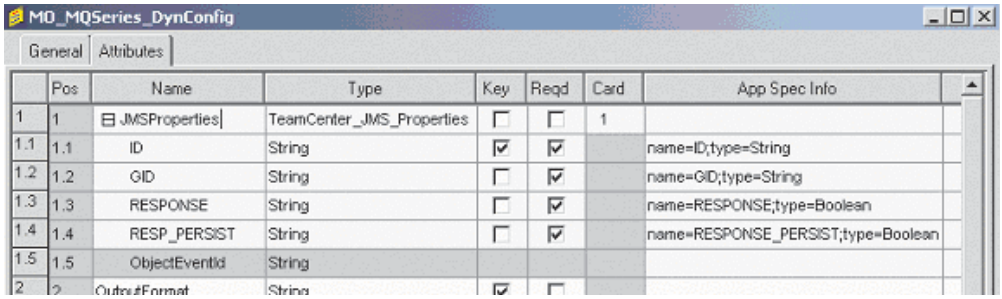

Figure 3. JMS properties attribute in a dynamic meta-object

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`evalue specified in the connector configuration properties. If you additionally specify a negative `ResponseTimeout` (meaning that the connector should not wait for a response), the`ReplyToQueue` 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 HL7, you must configure the startup file with the path information for the MQ Java client libraries. If the MQ Java client libraries are not listed in the startup file, use the instructions in the sections below to configure these files for Windows or UNIX systems.

Windows

To complete the configuration of the connector for Windows platforms, you must modify the `start_HL7.bat` file:

1. Open the `start_HL7.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_HL7.sh` file:

1. Open the `start_HL7.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.

Creating multiple connector instances

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

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

Create a new directory

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

```
ProductDir\connectors\connectorInstance
```

where `connectorInstance` uniquely identifies the connector instance.

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

```
ProductDir\repository\connectorInstance
```

Create business object definitions

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

1. If you need to modify business object definitions that are associated with the initial connector, copy the appropriate files and use Business Object Designer(?) to import them. You can copy any of the files for the initial connector. Just rename them if you make changes to them.

2. Files for the initial connector should reside in the following directory:

   `ProductDir\repository\initialConnectorInstance`

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

Create a connector definition

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

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

Create a start-up script

To create a startup script:

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

   dirname
2. Put this startup script in the connector directory you created in "Create a new directory" on page 44.
3. Create a startup script shortcut (Windows only).
4. Copy the initial connector's shortcut text and change the name of the initial connector (in the command line) to match the name of the new connector instance.

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

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

Starting the connector

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

ProductDir`\connectors\`*connName*

where *connName* identifies the connector. The name of the startup script depends on the operating-system platform, as Table 24 shows.

Table 24. Startup scripts for a connector

Operating system	Startup script
UNIX-based systems	`connector_manager_`*connName*
Windows	`start_`*connName*`.bat`

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

- On Windows systems, from the **Start** menu

  Select **Programs>IBM WebSphere Business Integration Adapters>Adapters>Connectors**. By default, the program name is "IBM

WebSphere Business Integration Adapters". However, it can be customized. Alternatively, you can create a desktop shortcut to your connector.

- From the command line
  - On Windows systems:

    `start_connName connName brokerName [-cconfigFile ]`
  - On UNIX-based systems:

    `connector_manager_connName -start`

  where *connName* is the name of the connector and *brokerName* identifies your integration broker, as follows:
  - For WebSphere InterChange Server, specify for *brokerName* the name of the ICS instance.
  - For WebSphere message brokers (WebSphere MQ Integrator, WebSphere MQ Integrator Broker, or WebSphere Business Integration Message Broker) or WebSphere Application Server, specify for *brokerName* a string that identifies the broker.

  Note: For a WebSphere message broker or WebSphere Application Server on a Windows system, you *must* include the `-c` option followed by the name of the connector configuration file. For ICS, the `-c` is optional.
- From Adapter Monitor (WebSphere Business Integration Adapters product only), which is launched when you start System Manager

  You can load, activate, deactivate, pause, shutdown or delete a connector using this tool.
- From System Monitor (WebSphere InterChange Server product only)

  You can load, activate, deactivate, pause, shutdown or delete a connector using this tool.
- On Windows systems, you can configure the connector to start as a Windows service. In this case, the connector starts when the Windows system boots (for an Auto service) or when you start the service through the Windows Services window (for a Manual service).

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

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

Stopping the connector

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

- If you started the connector from the command line, with its connector startup script:
  - On Windows systems, invoking the startup script creates a separate "console" window for the connector. In this window, type "Q" and press Enter to stop the connector.
  - On UNIX-based systems, connectors run in the background so they have no separate window. Instead, run the following command to stop the connector:

```
connector_manager_connName -stop
```

where *connName* is the name of the connector.

- From Adapter Monitor (WebSphere Business Integration Adapters product only), which is launched when you start System Manager

  You can load, activate, deactivate, pause, shutdown or delete a connector using this tool.
- From System Monitor (WebSphere InterChange Server product only)

  You can load, activate, deactivate, pause, shutdown or delete a connector using this tool.
- On Windows systems, you can configure the connector to start as a Windows service. In this case, the connector stops when the Windows system shuts down.

Chapter 3. Business objects

The connector for health care data protocols 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 health care data protocols, your modifications must conform to the rules the connector is designed to follow, or the connector cannot process new or modified business objects correctly.

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

Connector business object requirements

The business object requirements for the connector reflect the way the health care data handler converts:

- an HL7 message into a WebSphere business object, and vice versa
- a NCPDP message into a WebSphere business object, and vice versa

The sections below discuss the requirements for WebSphere business objects as well as the HL7 and NCPDP message structure.

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.

Overview of the HL7 message structure

HL7 standard supports primarily two modes of operations: Batch and Interactive. Data exchanged between systems can roughly be classified into two groups: individual and bulk to suit the two modes of operations.

By definition, bulk message is a collection of individual messages. It consists of header, sequentially listed individual message, and trailer. The embedded individual message retains its structure, just as it were in the interactive operation mode. There is no structural difference between individual messages operating in two different modes of operation. But because the nature of batch operation does not allow dialog messages, certain categories of individual message are only suited for the interactive operation mode.

Structurally all individual message contains a header. Some contains body and others don't. An Individual message is, according to the standard, an "atomic unit of data transferred between systems". However the very same standard also allows the notion of logical message, whose data is physically broken down to more than one individual messages and correlated together using a logical message id in

message headers. The breakup of a message into individual messages is driven primarily by message length negotiated between parties engaging in message exchanges.

For more information about the HL7 message structure and HL7 message components see

Overview of business objects for HL7

HL7 provides two ways by which to offer message representation:

- Sampled object representation
  The HL7 message body offers object representation to just the header. The message body is treated as a blob. One business object, to represent all message types.
- Native object representation
  This level of object representation of HL7 message offers object representation to both message header and message body. A corresponding business object represents virtually every HL7 data element, including every message structure or query.

The following sections describe rules to follow when constructing business objects to represent HL7 data elements.

Supported native message

When creating, configuring or modifying supported native messages:

- All message types and queries, except examples, defined by version 2.4 of HL7 message standard specification
- Industry specific business objects (ISBO) that correspond to the template used in the standard specification are provided for reference only. Not all data elements are fully solidified, thus no support is provided to these business objects.
- Earlier versions of the HL7 standard permited users to create their own custom made (CM) data type. Since CM data types are not specified in the standard, they are not included in the published collection of industry specific business objects (ISBOs). However, the meta-data model employed in the published ISBOs offers users a way to create their ASBO representation to these CM data types

Mapping the primitive data type

Use the following table to map the primitive data type:

Table 25. Mapping the Primitive data type

HL7 data elements	Business object attribute data type
SI	String
ID	String
NM	Float
DT	String
IS	String
ST	String
FT	String
TX	String

Mapping repeating data elements

Use the following table to map repeating data elements:

Table 26. Mapping repeating data elements

Repeating HL7 data element	BO attribute data type	Cardinality	ASI
SI, ID, DT, IS, ST, FT, TX	CW_Array_String	N	DataTypeID=HL7 data type ID
NM	CW_Array_Float	N	DataTypeID=NM

ISBO definitions

Documenting all of the business objects needed to represent the complete HL7 data element catalog is prohibitive. The following section explains how the business objects are constructed, along with noting several the special treatment required for several HL7 data elements hat do not follow traditional BO construction rules

Business object construction follows the HL7 data element structure very closely, except for the `HL7_DTEncodedText`, `HL7_DTMA`, `HL7_DTNA`, `HL7_DTQIP`, and `HL7_DTUnionALL`.

Note: The rules of constructing BO from HL7 data elements are detailed in the following subsections. The naming convention adopts [] to signify optionality and <> to signify the explanation of content enclosed by the angle brackets which is not part of the required characters in the naming convention. Each business object must contain the `ObjectEventID` BO. For this reason, the following business objects do not specifically list that business object.

BO Name

Message BO

Construct the Message BO using the following parameters:

- Corresponding HL7 data element
  Message structure identified by a pair of HL7 message type and event/query IDs
- Naming convention
  `HLT_MT<message type>_<Event code/query id>`
  where event code with smallest lexigraphical value is applicable to the message type/query
- Marking convention
  N/A
- Grammar
  N/A
- Example:
  `HL7_MTADT_A01` for ADT message type with ACK event with event code A01, A04, A08, and A13
  `HL7_MTQBP_Q21` for Query statement ID Q21 and Q24.

Group BO

The Group BO construct is summarized as follows:

- Corresponding HL7 data element
  Any segment group with two or more segments.
- Naming convention
  `HLT_MT<message type>_<Event code/query id>_GP<group index>`
  where event code with the smallest lexical value is applicable to the message type/query and index is unique in the scope of the message structure.
- Marking convention
  N/A
- Grammar
  N/A
- Example:
  `HL7_MTADT_A03` has a group name `HL7_MTADT_A03_GP1`

Table 27. Naming conventions for segment group

ADT^A03^ADT_A03	**ADT Message**
MSH	Message header
EVN	Event type
PID	Patient identification
[PD1]	Additional demographics
[{ ROL }]	Role
[PV2]	Patient visit - Additional information
[{ ROL }]	Role
[{ DB1 }]	Disability information
[{ DG1 }]	Diagnosis information
[DRG]	Diagnosis related group
[{	
PR1	Procedures
[{ ROL }]	Role
}]	
[{ OBX }]	Observation/Result
[PDA }	Patient death/autopsy

Segment BO

The construction of the Segment BO is summarized as follows:

- Corresponding HL7 data element
  HL7 segment and the segment definition of QPD, QED, RCP, QAK segments with their parameter tables of a conformance statement
- Naming convention
  `HL7_SG<segment ID>_[<Query ID>]`
  where Segment ID is the segment ID defined by HL7 and Query ID is the query ID defined by HL7 or user. Only QPD, QED, RCP, QAK segments require query ID.
- Marking convention
  N/A
- Grammar
  N/A

- Example:
  The segment BO for MSH is named HL7_SGMSHThe segment BO for QBD segment in QBP_Q21 is named HL7_SGQBD_Q21

Complex data type BO

The construction of a complex data type BO is summarized as follows:

- Corresponding HL7 data element
  Complex HL7 data type except MA, NA, and QIP data types
- Naming convention
  HL7_DT<data type>
  where Data Type is the data type ID defined by HL7 for complex HL7 data types.

  To the CE data type the following convention is used: HL7_DT<Data Type>_<Table Number>
  where table number is the id of table for the segment field of CE data type, which is defined in a segment definition

  The polymorphic nature of CM data type warrants separate treatment. Each instance of CM data type is specified in the following table. The data structure name is specified in the table of Data Structures in the HL7 Message Standard Access Database.

Table 28. List of all business objects for complex data types for CM data

DT BO	Data structure name	Description
HL7_DTAUI	AUI	Authorization information
HL7_DTCCD	CCD	Charge time
HL7_DTCCP	CCP	Channel calibration parameters
HL7_DTCSU	CSU	Channel sensitivity/units
HL7_DDI	DDI	Daily deductible
HL7_DIN	DIN	Activation code
HL7_DLD	DLD	Discharge location
HL7_DLT	DLT	Delta check
HL7_DTN	DTN	Day, type and number
HL7_DEIP	EIP	Parent order
HL7_DTELD	ELD	Error
HL7_DTLA1	LA1	Location with address information (variant one)
HL7_DTLA2	LA2	Location with address information (variant two)
HL7_DMOC	MOC	Charge to practice
HL7_DMSG	MSG	Message type
HL7_DNDL	NDL	Observing practitioner
HL7_DNR	NR	Wertebereich
HL7_DOCD_	OCD	Occurence
HL7_DOSD	OSD	Order sequence
HL7_DOSP	OSP	Occurence span
HL7_DPCF	PCF	Pre-certification
HL7_DPEN	PEN	Penalty
HL7_DTPI	PI	Personal identifier

Table 28. List of all business objects for complex data types for CM data (continued)

DT BO	Data structure name	Description
HL7_DTPIP	PIP	Privileges
HL7_DTPLN	PLN	Practitioner ID numbers
HL7_DTPRL	PRL	Parent result link
HL7_DTPTA	PTA	Policy type
HL7_DTRFR	RFR	Reference range
HL7_DTRMC	RMC	Room coverage
HL7_DTSPD	SPD	Specialty
HL7_DTSPS	SPS	Specimen source
HL7_DTUVC	UVC	Value code and amount
HL7_DTVR	VR	Value qualifier
HL7_DTWVI	WFI	Channel identifier
HL7_DTWVS	WVS	Waveform source

- Marking convention
  N/A
- Grammar
  N/A
- Example:
  The BO for CQ data type is named `HL_DTCQ`.
  The 46th field of OBR segment, which is of CE data type, uses table 0411, as shown the following excerpt. The BO name of this CE data type is `HL7_DTCE_0411`.

Table 29. 46th field of the OBR segment

SEQ	LEN	DT	OPT	RP/#	TBL#	ITEM #	Element Name
46	250	CE	O	Y	0411	01474	Placer supplemental service information

Data type union BO

The construct of the Data type union BO is summarized as follows:

- Corresponding HL7 data element
  Varies according to data type
- Naming convention
  `HL7_DTUnionAll`
- Marking convention
  N/A
- Grammar
  N/A
- Example:
  N/A

BO AppSpecificInfo

The following details application specific properties for common business objects.

Message BO

The construct of the Message BO for AppSpecificInfo is summarized as follows:

- Corresponding HL7 data element
  Message structure identified by a pair of HL7 message type and event /query ID
- Naming convention
  N/A
- Marking convention
  N/A
- Grammar
  N/A
- Example:
  N/A

Group BO

The construct of the Group BO for AppSpecificInfo is summarized as follows:

- Corresponding HL7 data element
  Any segment group with two or more segments.
- Naming convention
  N/A
- Marking convention
  N/A
- Grammar
  `StructType=Group`
  The values of `StructType` ASI property is defined in the following table:

Table 30. StructType value definition table

StructType value	Description
Group	To signify that the BO is for a segment group
Segment	To signify that the BO is for a segment
DataType	To signify that the BO is for an HL7 data type
Union	To signify that the BO is a union BO, which represents the polymorphic data type. Attributes of union BO represent data types permitted in the variant range of data types. Only one of the attributes in a union BO is populated at runtime. The context into how to choose valued attributed is indicated by Type Context ASI property
Array	To signify that the BO is for a component array container

- Example:
  N/A

Segment BO

The construct of the Segment BO for AppSpecificInfo is summarized as follows:

- Corresponding HL7 data element
  HL7 segment and the segment definition of QPD, QED, RCP, QAK segments with their parameter tables of a conformance statement.
- Naming convention
  N/A
- Marking convention
  N/A

- Grammar
  `StructType=Segment;SegID=<segment ID>`
  where segment ID is the HL7 segment ID
- Example:
  N/A

BO of a Complex data type

The construct of the BO for a complex data type for BO AppSpecificInfo is summarized as follows:

- Corresponding HL7 data element
  Complex HL7 data type except MA, NA and QIP data type
- Naming convention
  N/A
- Marking convention
  N/A
- Grammar
  N/A
- Example:
  N/A

Data type Union BO

The construct of the data type Union BO for BO AppSpecificInfo is summarized as follows:

- Corresponding HL7 data element
  Varies according to data type
- Naming convention
  N/A
- Marking convention
  N/A
- Grammar
  N/A
- Example:
  N/A

BO attribute structure

The following section details the attribute structure for several common business objects.

Message BO

The construct of the data type Message BO for BO Attribute Structure is summarized as follows:

- Corresponding HL7 data element
  Attribute of the types of segment BO or segment group BO. The actual list of attributes is specified by the HL7 message structure definition
- Naming convention
  N/A
- Marking convention
  N/A
- Grammar
  N/A

- Example:
  The message structure for message type ADT and event type A61 has the following attribute types:

Table 31. Attribute types for each attribute of HL7_MTADT_A61 and HL7 counter parts

Attribute type	Attribute type	HL7 Counterpart
1	HL7_SGMSH	MSH
2	HL7_SGEVN	EVN
3	HL7_SGPID	PID
4	HL7_SGPD1	[PD1]
5	HL7_SGV1	PV1
6	HL7_SGROL	[{ROL}]
7	HL7_SGPV2	[PV2}

Group BO

The construct of the data type Group BO for BO Attribute Structure is summarized as follows:

- Corresponding HL7 data element
  Any segment group with two or more segments
- Naming convention
  N/A
- Marking convention
  N/A
- Grammar
  N/A
- Example:
  BO `HL7_MTADT_A03_GP1` with HL7 counter parts depicted in the following table with its HL7 counter part.

Table 32. Attribute sequence for each attribute of HL7_MTADT_A03_GP1 and their HL7 counter parts

Attribute sequence	Attribute type	HL7 counterpart
1	HL7_SGPR1	PR1
2	HL7_SGROL	[{ROL}]

Segment BO

The construct of the data type Segment BO for BO AttributeStructure is summarized as follows:

- HL7 segment and the segment definition of QPD, RDT segments with their parameter tables of a conformance statement.
- Naming convention
  N/A
- Marking convention
  N/A
- Grammar
  N/A

- Example:
  BO HL7_SGQBD_Q21 with HL7 message standard specification definition for query Q21.
-

Table 33. Attribute sequence for query definition Q21 in BO HL7_SGQBD_Q21

Attribute sequence	Attribute type	HL7 counterpart
1	HL7_DTCE	MessageQueryName (CE)
2	HL7_DTENDCODEDTEXT	QueryTag (ST)
3	HL7_DTCX	PersonIdentifier (CX)
4	HL7_DTCX	WhatDomainsReturned (CSX)

BO of complex data type

The construct of the complex data structure for BO AttributeStructure is summarized as follows:

- Corresponding HL7 data element
  Any segment group with two or more segments
- Naming convention
  N/A
- Marking convention
  N/A
- Grammar
  N/A
- Example:
  BO HL7_MTADT_A03_GP1 with HL7 counter parts depicted in the following table with its HL7 counter part.

Table 34. Attribute sequence for each attribute of HL7_MTADT_A03_GP1 and their HL7 counter parts

Attribute sequence	Attribute type	HL7 counterpart
1	HL7_SGPR1	PR1
2	HL7_SGROL	[{ ROL }]

Data type Union BO

The Union BO represents the total aggregate of all data types defined by the standard specification.

Note: The Union BO must not recursively contain another Union BO.

The construct of the Union BO is summarized as follows:

- Corresponding HL7 data element
  - HL7 data type except CM and Varies
  - CM values from Table 28 on page 56
- Naming convention
  N/A
- Marking convention
  N/A

- Grammar
  N/A
- Example:
  N/A

BO Attribute Property Name

The following details the attribute property names for common business objects

Message BO

The property name attributes of the Message BO are summarized as follows:

- Corresponding HL7 data element
  Attributes in segment groups or segments that make up the HL7 message structure definition or query structure definition.
- Naming convention
  `<concantenation of words in description>`
  where the concatenation follows the following rules:

Table 35. Concatenation rules

Rule name	Rule description
Word selection	All words except the following punctuations and auxiliary words, regardless of their letter capitalization, in the description are included in the concatenation: ″a″, ″an″, ″the″, ″in″, ″of″, ″for″, ″with″, ″at″, ″/″, ″\″, ″-″, ″(″, ″)″,″[″, ″]″, ″'″, space, tab
Data type	Some descriptions contain parenthesis enclosing a data type id. These data types are stripped out along with the parenthesis.
Word capitalization	First letter of each remaining words is capitalized, and the remaining letters of the each remaining word are all suppressed to lower case.

Note: HL7 standard specification uses tables to convey the message structure or query message structure. However, even though each column contains brief descriptive words, none of the table columns bears the name “description” in the column title. That is to what “attribute description” refers. In the follow example, “ADT message” column is equivalent to the description column

- Marking convention
  N/A
- Grammar
  N/A
- Example:
  The following example uses BO `HL7_MTADT_A61`.

Table 36. Attribute names of BO HL7_MTADT_A61, constructed from the message structure of ADT_A61. ADT message column is the description of each attribute in the message structure definition

HL7 message structure ADT^A61^ADT^A61	ADT message	Message BO attribute name
MSH	Message handler	Message Header
EVN	Event type	Phenotype
PID	Patient identification	Patient Identification

Table 36. Attribute names of BO HL7_MTADT_A61, constructed from the message structure of ADT_A61. ADT message column is the description of each attribute in the message structure definition (continued)

HL7 message structure ADT^A61^ADT^A61	ADT message	Message BO attribute name
[PD1]	Additional demographics	Additional Demographics
PV1	Patient visit	Potentialities
[{ROL}]	Role	Role
[PV2}	Patient visit -- additional information	Patient Visit Additional Info

Group BO

The property name attributes of the Group BO are summarized as follows:

- Corresponding HL7 data element
  Segments or segment subgroup of a given segment group with two or more segments
- Naming convention
  `HLT_MT<message type>_<Event code/query id>_GP<group index>`
  where event code with the smallest lexigraphical value is applicable to the message type/query and group index is unique in the scope of the message structure. That is, the attribute name of Group BO is the same as its type
- Marking convention
  N/A
- Grammar
  N/A
- Example:
  The attribute name for the only group in BO `HL_MTADT_A03` is `HL_MTADT_A03`. See the table under the heading *Complex data types* for more information.

Segment BO

The property name attributes of the Segment BO are summarized as follows:

- Corresponding HL7 data element
  - The field names, or element name, of the regular HL7 segment
  - The parameter names of QPD, RDT segments of a conformance statement
- Naming convention
  `<concatenation of words in the element name of regular segment or the parameter name in the parameter table>`
  where event code with the smallest lexigraphical value is applicable to the message type/query and group index is unique in the scope of the message structure. That is, the attribute name of Group BO is the same as its type
- Marking convention
  N/A
- Grammar
  N/A
- Example:
  Using BO `HL_SGQBD_Q21`:

Table 37. The attribute names of HL7_SGQBD_Q21 and their HL7 counter parts

Attribute sequence	Segment BO attribute name	BO segment attribute type	HL7 counter part
1	MessageQueryName	HL7_DTCE	MessageQueryName (CE)
2	QueryTag	HL7_DTENDCODEDTEXT	QueryTag (ST)
3	PersonIdentifier	HL7_DTCX	PersonIdentifier
4	WhatDomains Returned	HL7_DTCX	WhatDomainsReturned

The segmented for MSH contains these values:

Table 38.

SEQ	MSH segment attribute name	Element name
1	Field separator	Field separator
2	Encoding characters	Encoding characters
3	Sending application	Sending application
4	Sending facility	Sending facility

BO of complex data type

The property name attributes of the complex data type BO are summarized as follows:

Corresponding HL7 data element
HL7 components in the definition data type except CM and Varies

- Naming convention
  `<concatenation of words in the component name ordinary data type or the description column of regular segment or the parameter name in the parameter table>`
  where the concatenation follows the rules listed in Table 35 on page 62. Additionally, the parenthesis at the end of the component, which used to indicate the component data type, are filtered out
- Marking convention
  N/A
- Grammar
  N/A
- Example:
  Using BO HL7_DTCP, which corresponds to the CP data type with the following description:
  `<price (MO)> ^ <price type (ID)> ^ <from value (NM)> ^ <to value (NM)> ^ <range units (CE)> ^ <range type (ID)>`

Table 39. Attribute names of BO complex data type HL7_DTCP as derived from the component names of the CP data type

Attribute name of HL7_DTCP	Component name of CP data type
Price	Price
PriceType	Price type
FromValue	From value
ToValue	To value
RangeUnits	Range units

Table 39. Attribute names of BO complex data type HL7_DTCP as derived from the component names of the CP data type (continued)

Attribute name of HL7_DTCP	Component name of CP data type
RangeType	Range type

Data type Union BO

The property name attributes of the Union BO are summarized as follows:

Corresponding to the HL7 data element

- HL7 data type except CM and Varies
- Data structure as listed in Table 28 on page 56
- Primitive data type
- Naming convention
  `<Name of complex data type>`
- Marking convention
  N/A
- Grammar
  N/A
- Example:
  The name of attribute of type `HL7_DTCP` is `HL7_DTCP` , the name of attribute NM is NM.

BO attribute property type

The following details the attribute property types for common business objects.

Message BO

The property type attributes of the Message BO are summarized as follows:

- Corresponding to the HL7 data element
  Attributes in segment groups or segments that make up the HL7 message structure definition or query structure definition.
- Naming convention
  `<BO names of corresponding segment group or segment>`
  Please see "Message BO" on page 58 for details on how BOs are named.
- Marking convention
  N/A
- Grammar
  N/A
- Example:
  See BO `HL7_MTADT_A61`, Table 36 on page 62, for the names of the attribute types.

Group BO

The property type attributes of the Group BO are summarized as follows:

- Corresponding to the HL7 data element
  Segments or segment subgroup of a given segment group with two or more segments
- Naming convention
  `<Name of segment BO in a segment group>`
  see "Message BO" on page 54for a complete list of

- Marking convention
  N/A
- Grammar
  N/A
- Example:
  See BO HL7_MTADT_A61, Table 36 on page 62.

Segment BO

The property type attributes of the Segment BO are summarized as follows:

- Corresponding to the HL7 data element
  - The fields of regular HL7 segment
  - The parameter names of QPD, RDT segments of a conformance statement
- Naming convention
  <Name of BO of complex data type or the appropriate primitive data type >
- Marking convention
  N/A
- Grammar
  N/A
- Example:
  See Table 37 on page 64.

BO of complex data type

The property type attributes of the complex data type are summarized as follows:

- Corresponding to the HL7 data element
  - The fields of regular HL7 segment
  - Components of data structures listed in Table 28 on page 56of this document for CM type
- Naming convention
  <Name of BO of complex data type that corresponds to the subconponent or the appropriate primitive data type >
- Marking convention
  N/A
- Grammar
  N/A
- Example:
  See Table 31 on page 60.

Data type Union BO

The property type attributes of the Union BO data type are summarized as follows:

- Corresponding to the HL7 data element
  - HL7 data type except CM and Varies
  - Components of data structures listed in Table 28 on page 56 of this document for CM type
  - Primitive data types
- Naming convention
  <Name of BO of complex data type that corresponds to the subconponent or the appropriate primitive data type >
- Marking convention
  N/A

- Grammar
  N/A
- Example:
  The name of attribute of type`HL7_DTCP` is `HL7_DTCP`; the name of attribute of type NM is NM

BO attribute property Iskey

The following details the attribute property Iskey for common business objects.

Message BO

Key attributes in the message BO bear no significance.

- Corresponding to the HL7 data element
  Attributes in segment groups or segments that make up the HL7 message structure definition or query structure definition.
- Naming convention
  N/A
- Marking convention
  Set key to true for the first attribute of the message BO.
- Example:
  N/A

Group BO

Key attributes in the group BO bear no significance.

- Corresponding to the HL7 data element
  Segments or segment subgroup of a given segment group with two or more segments
- Naming convention
  N/A
- Marking convention
  Set key to true for the first attribute of the group BO.
- Example:
  N/A

Segment BO

Key attributes in Segment BO are derived from ordinary segments and do not bear any significance. Their presence is only to fulfill the architecture's system requirements, but are crucial to parameters in the QPD, RDT segments

- Corresponding to the HL7 data element
- The fields of regular HL7 segment
- The parameter names of QPD, RDT segments of a conformance statement
- Naming convention
  N/A
- Marking convention
  - For all segment BO except segments of type QPD, RDT, set the first attribute key to true
  - For segments of type QPD, RDT, set key to true for attributes whose corresponding parameter is marked as "Key"
- Grammar
  N/A

- Example:
  N/A

BO of complex data type

Key attributes in BO of Complex Data Type do not bear any significance.

- Corresponding to the HL7 data element
  - HL7 components in the definition data type except CM and Varies
  - Components of data structures listed in Table 28 on page 56of this document for CM type
- Naming convention
  N/A
- Marking convention
  - All BO of complex data type, including those listed in Table 28 on page 56and primitive type attributes
  - Set the IsKey property of the first attribute to true.
- Grammar
  N/A
- Example:
  N/A

BO attribute property IsForeignKey

The following details the attribute property IsKey for common business objects.

Message BO

Key attributes in the message BO bear no significance.

- Corresponding to the HL7 data element
  Attributes in segment groups or segments that make up the HL7 message structure definition or query structure definition.
- Naming convention
  N/A
- Marking convention
  Set IsForeignKey to false
- Grammar
  N/A
- Example
  N/A

Group BO

- Corresponding to the HL7 data element
  Segments or segment subgroup of a given segment group with two or more segments
- Naming convention
  N/A
- Marking convention
  Set IsForeignKey to false
- Example:
  N/A

Segment BO

- Corresponding to the HL7 data element
  - The fields of regular HL7 segment
  - The parameter names of QPD, RDT segments of a conformance statement
- Naming convention
  N/A
- Marking convention
  Set IsForeignKey to false
- Grammar
  N/A
- Example:
  N/A

BO of complex data type

Key attributes in BO of Complex Data Type do not bear any significance.

- Corresponding to the HL7 data element
-
  - HL7 components in the definition data type except CM and Varies
  - Components of data structures listed in Table 28 on page 56of this document for CM type
- Naming convention
  N/A
- Marking convention
  Set IsForeignKey to false
- Grammar
  N/A
- Example:
  N/A

Data type Union BO

The property type attributes of the Union BO data type are summarized as follows:

- Corresponding to the HL7 data element
  - HL7 data type except CM and Varies
  - Components of data structures listed in Table 28 on page 56 of this document for CM type
  - Primitive data types
- Naming convention
  N/A
- Marking convention
  Set IsForeignKey to false
- Grammar
  N/A
- Example:
  N/A

BO attribute property Cardinality

The following details the attribute property Cardinality for common business objects.

Message BO

Key attributes in the message BO bear no significance.

- Corresponding to the HL7 data element
  Attributes in segment groups or segments that make up the HL7 message structure definition or query structure definition.
- Naming convention
  N/A
- Marking convention
  Segment group or segment that has {} brace: set cardinality to N else to 1
- Example:
  The corresponding message BO for the ADT_A61 HL7 message structure for message type ADT and event type A61 has following attributes:

Table 40. This example shows the attribute cardinality for each attribute of HL7_MTADT_A61 and their HL7 counter parts

Attribute sequence	Attribute type	HL7 counter part	Cardinality
1	HL7_SGMSH	MSH	1
2	HL7_SGEVN	EVN	1
3	HL7_SGPID	PID	1
4	HL7_SGPD1	[PD1]	1
5	HL7_SGV1	PV1	1
6	HL7_SGROL	[{ ROL }]	N
7	HL7_SGPV2	[PV2]	1

Group BO

- Corresponding to the HL7 data element
  Segments or segment subgroup of a given segment group with two or more segments
- Naming convention
  N/A
- Marking convention
  Set IsForeignKey to false
- Example:
  N/A

Segment BO

- Corresponding to the HL7 data element
  - The fields of regular HL7 segment
  - The parameter names of QPD, RDT segments of a conformance statement
- Naming convention
  N/A
- Marking convention
  Segment fields that have the RP# or RP column marked with ″Y″ or ″y″: set cardinality to N else set to 1
- Grammar
  N/A

- Example:
  For BO HL7_SGQBD_Q21, the cardinality attribute for the HL7 message standard specification for parameter definition for query Q21:

Table 41. This example shows the attribute cardinality for each attribute of HL7_SGQBD_Q21 and their HL7 counter parts

Attribute sequence	Sequence BO attribute name	RP#/RP	Cardinality
1	MessageQueryName		1
2	QueryTag		1
3	PersonIdentifier	N	1
4	WhatDomainReturned	Y	N

BO of complex data type

Key attributes in BO of Complex Data Type do not bear any significance.

- Corresponding to the HL7 data element
  - HL7 components in the definition data type except CM and Varies
  - Components of data structures listed in Table 28 on page 56of this document for CM type
- Naming convention
  N/A
- Marking convention
  Set cardinality to 1
- Grammar
  N/A
- Example
  N/A

Data type Union BO

The property type attributes of the Union BO data type are summarized as follows:

- Corresponding to the HL7 data element
  - HL7 data type except CM and Varies
  - Components of data structures listed in Table 28 on page 56 of this document for CM type
  - Primitive data types
- Naming convention
  N/A
- Marking convention
  Set cardinality to 1
- Grammar
  N/A
- Example:
  N/A

BO attribute property MaxLength

The following details the attribute property MaxLength for common business objects.

Because the architecture BO does not have the notion of BO size, a characteristic strongly exhibited in all HL7 data elements instead of just data element of primitive type, the notion of Maximum Length can't be conveyed in the traditional manner through the MaxLength BO attributes property. This attribute property is not used in BOs of all scopes of HL7 data element representation in order to achieve uniformity in the representation of Maximum Length of HL7 data element. Instead, the notion of Maximum Length of HL7 data element is conveyed through the AppSpecificInfo. Since this version of the data handler does not support validation, Maxlength as an AppSpecificInfo property does not take place in any part of the BO of this release.

Since the MaxLength can be a decisive factor on whether BOs of certain size can be stored in communication channels for certain brokers, it is still important to set the correct value to this attribute property. If it is not mentioned, the default value of this attribute property is 255 for all primitive attributes.

BO attribute property IsRequired

The following details the attribute property IsRequired for common business objects.

Message BO

- Corresponding to the HL7 data element
  Attributes in segment groups or segments that make up the HL7 message structure definition or query structure definition.
- Naming convention
  N/A
- Marking convention
  Set IsRequired to false if the HL7 message structure definition encloses the segment or segment group with square bracket [], otherwise set the property to true.
- Example:
  The corresponding message BO for the ADT_A61 HL7 message structure for message type ADT and event type A61 has following attributes:

Table 42. This example shows the attribute cardinality for each attribute of HL7_MTADT_A61 and their HL7 counter parts

Attribute sequence	Attribute type	HL7 counter part	IsRequired
1	HL7_SGMSH	MSH	True
2	HL7_SGEVN	EVN	True
3	HL7_SGPID	PID	True
4	HL7_SGPD1	[PD1]	False
5	HL7_SGV1	PV1	True
6	HL7_SGROL	[{ ROL }]	False
7	HL7_SGPV2	[PV2]	False

Group BO

- Corresponding to the HL7 data element
  Segments or segment subgroup of a given segment group with two or more segments

- Naming convention
  N/A
- Marking convention
  Segment group or segment BO that has [] brace: set IsRequired to false, otherwise set to true
- Example:
  N/A

Segment BO

- Corresponding to the HL7 data element
  - The fields of regular HL7 segment
  - The parameter names of QPD, RDT segments of a conformance statement
- Naming convention
  N/A
- Marking convention
  If the OPT column of segment field or query parameter is not set to O, then set IsRequired property of corresponding attribute to false, otherwise set to true.
- Grammar
  N/A
- Example:
  For BO `HL7_SGQBD_Q21`, the IsRequired attribute for the HL7 message standard specification for parameter definition for query Q21:

Table 43. This example shows the IsRequired attribute for HL7_SGQBD_Q21 and their HL7 counter parts

Attribute sequence	Sequence BO attribute name	OPT	IsRequired
1	MessageQueryName	R	True
2	QueryTag	R	True
3	PersonIdentifier	R	True
4	WhatDomainReturned	O	False

BO of complex data type

Key attributes in BO of Complex Data Type do not bear any significance.

- Corresponding to the HL7 data element
  - HL7 components in the definition data type except CM and Varies
  - Components of data structures listed in Table 28 on page 56of this document for CM type
- Naming convention
  N/A
- Marking convention
  Set IsRequired to false
- Grammar
  N/A
- Example:
  N/A

Data type Union BO

The property type attributes of the Union BO data type are summarized as follows:

- Corresponding to the HL7 data element
  - HL7 data type except CM and Varies
  - Components of data structures listed in Table 28 on page 56 of this document for CM type
  - Primitive data types
- Naming convention
  N/A
- Marking convention
  Set IsRequired to false
- Grammar
  N/A
- Example:
  N/A

BO attribute property Relationship

The following details the attribute property Relationship for common business objects.

Message BO

- Corresponding to the HL7 data element
  Attributes in segment groups or segments that make up the HL7 message structure definition or query structure definition.
- Naming convention
  N/A
- Marking convention
  Set Relationship to Containment
- Example:
  N/A

Group BO

- Corresponding to the HL7 data element
  Segments or segment subgroup of a given segment group with two or more segments
- Naming convention
  N/A
- Marking convention
  Set Relationship to Containment
- Example:
  N/A

Segment BO

- Corresponding to the HL7 data element
  - The fields of regular HL7 segment
  - The parameter names of QPD, RDT segments of a conformance statement
- Naming convention
  N/A

- Marking convention
  Set Relationship to Containment
- Grammar
  N/A
- Example:
  N/A

BO of complex data type

Key attributes in BO of Complex Data Type do not bear any significance.

- Corresponding to the HL7 data element
  - HL7 components in the definition data type except CM and Varies
  - Components of data structures listed in Table 28 on page 56of this document for CM type
- Naming convention
  N/A
- Marking convention
  If the attribute type is BO, set Relationship to Containment, otherwise take out the attribute property when working with BO definition file
- Grammar
  N/A
- Example:
  N/A

Data type Union BO

The property type attributes of the Union BO data type are summarized as follows:

- Corresponding to the HL7 data element
  - HL7 data type except CM and Varies
  - Components of data structures listed in Table 28 on page 56 of this document for CM type
  - Primitive data types
- Naming convention
  N/A
- Marking convention
  If the attribute type is BO, set Relationship to Containment, otherwise take out the attribute property when working with BO definition file
- Grammar
  N/A
- Example:
  N/A

BO attribute property AppSpecificInfo

The following details the attribute property Relationship for common business objects.

Message BO

- Corresponding to the HL7 data element
  Attributes in segment groups or segments that make up the HL7 message structure definition or query structure definition.

- Naming convention
  N/A
- Marking convention
  N/A
- Example:
  N/A

Group BO

- Corresponding to the HL7 data element
  Segments or segment subgroup of a given segment group with two or more segments
- Naming convention
  N/A
- Marking convention
  N/A
- Example:
  N/A

Segment BO

- Corresponding to the HL7 data element
  - The fields of regular HL7 segment
  - The parameter names of QPD, RDT segments of a conformance statement
- Naming convention
  N/A
- Marking Convention
  N/A
- Grammar
  - All data types other than CM: `StructType=DataType;DataTypeID=<HL7 data type ID>`
  - DT BO listed in Table 28 on page 56`StructType=DataType;DataTypeID=CM`
- Example:
  - For attribute that corresponds to the ST data type, the AppSpecificInfo is `StructType=DataType;DataTypeID=ST; MaxLength=199`
  - For attribute whose type is HL7_DTMSG as defined in Table 28 on page 56, the AppSpecificInfo is `StructType=DataType;DataTypeID=CM`

BO of complex data type

Key attributes in BO of Complex Data Type do not bear any significance.

- Corresponding to the HL7 data element
  - HL7 components in the definition data type except CM and Varies
  - Components of data structures listed in Table 28 on page 56of this document for CM type
- Naming convention
  N/A
- Marking convention
  If the attribute type is BO, set Relationship to Containment, otherwise take out the attribute property when working with BO definition file
- Grammar

- All data types other than CM, MA, NA, QIP and Varies
  `StructType=DataType;DataTypeID=<HL7 data type ID>`
- DT BO listed in Table 28 on page 56, `StructType=DataType;DataTypeID=CM`
- MA, NA and QIP type BO
- Union BO for varies data type `StructType=Union;TypeContext=<reference path at where the data type is specified in the BO structure>`

TypeContext is used to indicate where in the BO structure to retrieve the data type announcer information such as the CM or NM for the data carrier.

The path can be relative or can also be absolute path where the root is the top-most parent BO.

Since the RDT segment definition expects user to provide definition for each of the columns and hence does not consist of column of varies data type, the need to supporting absolute path is not warranted. This release only supports the relative path.

In this release, the relative pathname is defined as BO attribute name of the one of data type announcer

- Example:

Table 44. This example shows the attribute AppSpecificInfo for each attribute of HL7_DTCQ and their HL7 counter parts

Attribute sequence	Attribute name	Attribute type	ASI
1	Quantity	Float	DataTypeID=NM
2	Units	HL7_DTCE	DataTypeID=CM

The ObservationValue attribute of BO `HL7_SGOBX`, which corresponds to the OBX-5 (varies) and is depended on the data type announcer located at OBX-2, has AppSpecificInfo `StructType=Union;TypeContext=ValueType`

Here the path to the data type announcer is ″ValueType″, which is just the attribute name. Because both the ″ValueType″ and ″ObservationValue″ attributes are located in the same BO, this path is a relative path.

Data type Union BO

The property type attributes of the Union BO data type are summarized as follows:

- Corresponding to the HL7 data element
  - HL7 data type except CM and Varies
  - Components of data structures listed in Table 28 on page 56 of this document for CM type
  - Primitive data types
- Naming convention
  N/A
- Marking convention
  N/A
- Grammar
  - All Data type other than CM `StructType=DataType;DataTypeID=<HL7 data type ID>`

– CM type BO listed in Table 28 on page 56`StructType=DataType;DataTypeID=CM`
– MA, NA type BO listed in Table 28 on page 56
`StructType=Array;DataTypeID=<MA or NA, whichever appropriate data type ID>`

• Example:

For attribute that corresponds to the ST data type, the AppSpecificInfo is `StructType=DataType;DataTypeID=ST`For an attribute whose type is HL7_DTMSG as defined in Table 28 on page 56, the AppSpecificInfo is `StructType=DataType;DataTypeID=CM`

HL7 Business Objects

The two included sample health care business objects are:

• BIA_MO_DataHandler_Healthcare.txt
• BIA_MO_DataHandler_Hl7.txt

BIA_MO_DataHandler_Healthcare.txt

The BIA_MO_DataHandler_Health.txt is located in the WebSphereBI/connectors/Healthcare/samples/ folder.

```
[ReposCopy]
Version = 3.0.0
[End]

[BusinessObjectDefinition]
Name = BIA_MO_DataHandler_Healthcare
Version = 1.0.0

[Attribute]
Name = ncpdp
Type = BIA_MO_DataHandler_NCPDP
ContainedObjectVersion = 3.0.0
Relationship = Containment
Cardinality = 1
MaxLength =
IsKey = true
IsForeignKey = false
IsRequired = false
IsRequiredServerBound = false
[End]

[Attribute]
Name = hl7
Type = BIA_MO_DataHandler_HL7
ContainedObjectVersion = 3.0.0
Relationship = Containment
Cardinality = 1
MaxLength =
IsKey = false
IsForeignKey = false
IsRequired = false
IsRequiredServerBound = false
[End]

[Attribute]
Name = ObjectEventId
Type = String
Cardinality = 1
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]
```

BIA_MO_DataHandler_HL7.txt

The BIA_MO_DataHandler_HL7.txt is located in the WebSphereBI/connectors/Healthcare/samples/ folder.

```
[ReposCopy] Version = 3.0.0 [End] [BusinessObjectDefinition] Name =
BIA_MO_DataHandler_HL7 Version = 3.0.0 [Attribute] Name = ClassName Type =
String Cardinality = 1 MaxLength = 255 IsKey = false IsForeignKey = false
IsRequired = false DefaultValue =
com.ibm.adapters.datahandlers.hl7.HL7DataHandler IsRequiredServerBound =
false [End] [Attribute] Name = BOPrefix Type = String Cardinality = 1
MaxLength = 255 IsKey = false IsForeignKey = false IsRequired = false
DefaultValue = HL7 IsRequiredServerBound = false [End] [Attribute] Name =
Representation Type = String Cardinality = 1 MaxLength = 255 IsKey = false
IsForeignKey = false IsRequired = false DefaultValue = simplified
IsRequiredServerBound = false [End] [Attribute] Name = FieldDelimiter Type
= String Cardinality = 1 MaxLength = 255 IsKey = false IsForeignKey = false
IsRequired = false DefaultValue = | IsRequiredServerBound = false [End]
[Attribute] Name = ComponentDelimiter Type = String Cardinality = 1
MaxLength = 255 IsKey = false IsForeignKey = false IsRequired = false
DefaultValue = ^ IsRequiredServerBound = false [End] [Attribute] Name =
RepetitionDelimiter Type = String Cardinality = 1 MaxLength = 255 IsKey =
true IsForeignKey = false IsRequired = false DefaultValue = ~
IsRequiredServerBound = false [End] [Attribute] Name = EscapeDelimiter Type
= String Cardinality = 1 MaxLength = 255 IsKey = true IsForeignKey = false
IsRequired = false DefaultValue = \ IsRequiredServerBound = false [End]
[Attribute] Name = SubcomponentDelimiter Type = String Cardinality = 1
MaxLength = 255 IsKey = false IsForeignKey = false IsRequired = false
DefaultValue = & IsRequiredServerBound = false [End] [Attribute] Name =
MTEventMap Type = String Cardinality = 1 MaxLength = 255 IsKey = false
IsForeignKey = false IsRequired = false DefaultValue =
file=C:\x\WebSphereAdapters\connectors\Healthcare\dependencies\
hl7\BIA_HL7MTEventMap.cfg IsRequiredServerBound = false [End] [Attribute]
Name = I18N Type = String Cardinality = 1 MaxLength = 255 IsKey = false
IsForeignKey = false IsRequired = false DefaultValue =
file=C:\x\WebSphereAdapters\connectors\Healthcare\dependencies\
hl7\BIA_HL7I18N.cfg IsRequiredServerBound = false [End] [Attribute] Name =
DummyKey Type = String Cardinality = 1 MaxLength = 255 IsKey = false
IsForeignKey = false IsRequired = false DefaultValue = dummy
IsRequiredServerBound = false [End] [Attribute] Name = DefaultVerb Type =
String Cardinality = 1 MaxLength = 255 IsKey = false IsForeignKey = false
IsRequired = false DefaultValue = Create IsRequiredServerBound = false
[End] [Attribute] Name = EnableStackTrace Type = String Cardinality = 1
MaxLength = 255 IsKey = false IsForeignKey = false IsRequired = false
```

```
DefaultValue = true IsRequiredServerBound = false [End] [Attribute] Name =
ObjectEventId Type = String Cardinality = 1 MaxLength = 255 IsKey = false
IsForeignKey = false IsRequired = false IsRequiredServerBound = false [End]
[Verb] Name = Create [End] [Verb] Name = Delete [End] [Verb] Name =
Retrieve [End] [Verb] Name = Update [End] [End]
```

Chapter 4. Troubleshooting

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

Startup problems

Problem	Potential solution / explanation
The connector shuts down unexpectedly during initialization and the following message is reported: `Exception in thread "main" java.lang.NoClassDefFoundError: javax/jms/JMSException...`	Connector cannot find file `jms.jar` from the IBM WebSphere MQ Java client libraries. Ensure that variable `MQSERIES_JAVA_LIB` in `start_connector.bat` points to the IBM WebSphere MQ Java client library folder.
The connector shuts down unexpectedly during initialization and the following message is reported: `Exception in thread "main" java.lang.NoClassDefFoundError: com/ibm/mq/jms/MQConnectionFactory...`	Connector cannot find file `com.ibm.mqjms.jar` in the IBM WebSphere MQ Java client libraries. Ensure that variable `MQSERIES_JAVA_LIB` in `start_connector.bat` points to the IBM WebSphere MQ Java client library folder.
The connector shuts down unexpectedly during initialization and the following message is reported: `Exception in thread "main" java.lang.NoClassDefFoundError: javax/naming/Referenceable...`	Connector cannot find file `jndi.jar` from the IBM WebSphere MQ Java client libraries. Ensure that variable `MQSERIES_JAVA_LIB` in `start_connector.bat` points to the IBM WebSphere MQ Java client library folder.
The connector 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 (`mqjbnd01.dll` [NT] or `libmqjbnd01.so` [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: `HostName`, `Channel`, and `Port`.

Event processing

Problem	Potential solution / explanation
The connector delivers all messages with an MQRFH2 header.	To deliver messages with only the MQMD WebSphere MQ header, append `?targetClient=1` to the name of output queue URI. For example, if you output messages to queue `queue://my.queue.manager/OUT`, change the URI to `queue://my.queue.manager/OUT?targetClient=1`. See Chapter 2, "Configuring the connector," on page 15 for more information.
The connector truncates all message formats to 8 characters upon delivery regardless of how the format has been defined in the connector meta-object.	This is a limitation of the WebSphere MQ MQMD message header and not the connector.

Appendix A. Standard configuration properties for connectors

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

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

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

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

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

New and deleted properties

These standard properties have been added in this release.

New properties

- XMLNameSpaceFormat

Deleted properties

- RestartCount
- RHF2MessageDomain

Configuring standard connector properties

Adapter connectors have two types of configuration properties:

- Standard configuration properties
- Connector-specific configuration properties

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

Using Connector Configurator

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

Note: Connector Configurator and System Manager run only on the Windows system. If you are running the connector on a UNIX system, you must have

a Windows machine with these tools installed. To set connector properties for a connector that runs on UNIX, you must start up System Manager on the Windows machine, connect to the UNIX integration broker, and bring up Connector Configurator for the connector.

Setting and updating property values

The default length of a property field is 255 characters.

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

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

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

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

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

Summary of standard properties

Table 45 on page 85 provides a quick reference to the standard connector configuration properties. Not all the connectors make use of all these properties, and property settings may differ from integration broker to integration broker, as standard property dependencies are based on `RepositoryDirectory`.

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

Table 45. Summary of standard configuration properties

Property name	Possible values	Default value	Update method	Notes
AdminInQueue	Valid JMS queue name	*CONNECTORNAME* /ADMININQUEUE	Component restart	Delivery Transport is JMS
AdminOutQueue	Valid JMS queue name	*CONNECTORNAME*/ADMINOUTQUEUE	Component restart	Delivery Transport is JMS
AgentConnections	1-4	1	Component restart	Delivery Transport is MQ or IDL: Repository directory is <REMOTE>
AgentTraceLevel	0-5	0	Dynamic	
ApplicationName	Application name	Value specified for the connector application name	Component restart	
BrokerType	ICS, WMQI, WAS			
CharacterEncoding	ascii7, ascii8, SJIS, Cp949, GBK, Big5, Cp297, Cp273, Cp280, Cp284, Cp037, Cp437 **Note:** This is a subset of supported values.	ascii7	Component restart	
ConcurrentEventTriggeredFlows	1 to 32,767	1	Component restart	Repository directory is <REMOTE>
ContainerManagedEvents	No value or JMS	No value	Component restart	Delivery Transport is JMS
ControllerStoreAndForwardMode	true or false	True	Dynamic	Repository directory is <REMOTE>
ControllerTraceLevel	0-5	0	Dynamic	Repository directory is <REMOTE>
DeliveryQueue		*CONNECTORNAME*/DELIVERYQUEUE	Component restart	JMS transport only
DeliveryTransport	MQ, IDL, or JMS	JMS	Component restart	If Repository directory is local, then value is JMS only
DuplicateEventElimination	True or False	False	Component restart	JMS transport only: Container Managed Events must be <NONE>
FaultQueue		*CONNECTORNAME*/FAULTQUEUE	Component restart	JMS transport only

Table 45. Summary of standard configuration properties (continued)

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	Component restart	JMS transport only
jms.MessageBrokerName	If FactoryClassName is IBM, use crossworlds.queue. manager. If FactoryClassName is Sonic, use localhost:2506.	crossworlds.queue.manager	Component restart	JMS transport only
jms.NumConcurrentRequests	Positive integer	10	Component restart	JMS transport only
jms.Password	Any valid password		Component restart	JMS transport only
jms.UserName	Any valid name		Component restart	JMS transport only
JvmMaxHeapSize	Heap size in megabytes	128m	Component restart	Repository directory is <REMOTE>
JvmMaxNativeStackSize	Size of stack in kilobytes	128k	Component restart	Repository directory is <REMOTE>
JvmMinHeapSize	Heap size in megabytes	1m	Component restart	Repository directory is <REMOTE>
ListenerConcurrency	1- 100	1	Component restart	Delivery Transport must be MQ
Locale	en_US, ja_JP, ko_KR, zh_CN, zh_TW, fr_FR, de_DE, it_IT, es_ES, pt_BR **Note:** This is a subset of the supported locales.	en_US	Component restart	
LogAtInterchangeEnd	True or False	False	Component restart	Repository Directory must be <REMOTE>
MaxEventCapacity	1-2147483647	2147483647	Dynamic	Repository Directory must be <REMOTE>
MessageFileName	Path or filename	InterchangeSystem.txt	Component restart	
MonitorQueue	Any valid queue name	*CONNECTORNAME*/MONITORQUEUE	Component restart	JMS transport only: DuplicateEvent Elimination must be True
OADAutoRestartAgent	True or False	False	Dynamic	Repository Directory must be <REMOTE>

Table 45. Summary of standard configuration properties (continued)

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

Table 45. Summary of standard configuration properties (continued)

Property name	Possible values	Default value	Update method	Notes
SynchronousRequestTimeout	0 - any number (millisecs)	0	Component restart	Delivery Transport is JMS
SynchronousResponseQueue		*CONNECTORNAME*/SYNCHRONOUSRESPONSEQUEUE	Component restart	Delivery Transport is JMS
WireFormat	CwXML, CwBO	CwXML	Agent restart	CwXML if Repository Directory is not <REMOTE>: CwBO if Repository Directory is <REMOTE>
WsifSynchronousRequest Timeout	0 - any number (millisecs)	0	Component restart	WAS only
XMLNameSpaceFormat	short, long	short	Agent restart	WebSphere MQ message brokers and WAS only

Standard configuration properties

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

AdminInQueue

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

The default value is CONNECTORNAME/ADMININQUEUE.

AdminOutQueue

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

The default value is CONNECTORNAME/ADMINOUTQUEUE.

AgentConnections

Applicable only if RepositoryDirectory is <REMOTE>.

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

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

AgentTraceLevel

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

ApplicationName

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

BrokerType

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

CharacterEncoding

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

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

By default, a subset of supported character encodings only is displayed in the drop 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.

ConcurrentEventTriggeredFlows

Applicable only if `RepositoryDirectory` is <REMOTE>.

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

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

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

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

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

ContainerManagedEvents

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

The default value is No value.

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

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

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

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

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

ControllerStoreAndForwardMode

Applicable only if RepositoryDirectory is <REMOTE>.

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

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

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

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

The default is true.

ControllerTraceLevel

Applicable only if RepositoryDirectory is <REMOTE>.

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

DeliveryQueue

Applicable only if DeliveryTransport is JMS.

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

The default value is CONNECTORNAME/DELIVERYQUEUE.

DeliveryTransport

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

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

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

WebSphere MQ and IDL

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

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

JMS

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

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

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

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

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

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

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

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

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

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

DuplicateEventElimination

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

This property can also be set to `false`.

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

FaultQueue

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

The default value is `CONNECTORNAME/FAULTQUEUE`.

JvmMaxHeapSize

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

The default value is 128m.

JvmMaxNativeStackSize

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

The default value is 128k.

JvmMinHeapSize

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

The default value is 1m.

jms.FactoryClassName

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

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

jms.MessageBrokerName

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

The default is `crossworlds.queue.manager`.

jms.NumConcurrentRequests

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

The default value is 10.

jms.Password

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

There is no default.

jms.UserName

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

There is no default.

ListenerConcurrency

This property supports multi-threading in MQ Listener when ICS is the integration broker. It enables batch writing of multiple events to the database, thus improving system performance. The default value is 1.

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

Locale

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

A locale name has the following format:

ll_TT.codeset

where:

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

By default, only a subset of supported locales appears in the drop list. To add other supported values to the drop list, you must manually modify the

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

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

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

LogAtInterchangeEnd

Applicable only if RespositoryDirectory is <REMOTE>.

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

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

MaxEventCapacity

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

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

MessageFileName

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

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

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

MonitorQueue

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

The default value is CONNECTORNAME/MONITORQUEUE

OADAutoRestartAgent

Valid only when the RepositoryDirectory is <REMOTE>.

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

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

The default value is `false`.

OADMaxNumRetry

Valid only when the `RepositoryDirectory` is <REMOTE>.

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

The default value is `1000`.

OADRetryTimeInterval

Valid only when the `RepositoryDirectory` is <REMOTE>.

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

The default is `10`.

PollEndTime

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

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

PollFrequency

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

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

The default is `10000`.

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

PollQuantity

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

PollStartTime

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

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

RequestQueue

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

The default value is `CONNECTOR/REQUESTQUEUE`.

RepositoryDirectory

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

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

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

ResponseQueue

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

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

RestartRetryCount

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

The default is `3`.

RestartRetryInterval

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

The default is 1.

SourceQueue

Applicable only if DeliveryTransport is JMS and ContainerManagedEvents is specified.

Designates the JMS source queue for the connector framework in support of guaranteed event delivery for JMS-enabled connectors that use a JMS event store. For further information, see "ContainerManagedEvents" on page 89.

The default value is CONNECTOR/SOURCEQUEUE.

SynchronousRequestQueue

Applicable only if DeliveryTransport is JMS.

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

The default is CONNECTORNAME/SYNCHRONOUSREQUESTQUEUE

SynchronousResponseQueue

Applicable only if DeliveryTransport is JMS.

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

The default is CONNECTORNAME/SYNCHRONOUSRESPONSEQUEUE

SynchronousRequestTimeout

Applicable only if DeliveryTransport is JMS.

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

The default value is 0.

WireFormat

Message format on the transport.

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

WsifSynchronousRequest Timeout

WAS integration broker only.

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

The default value is 0.

XMLNameSpaceFormat

WebSphere message brokers and WAS integration broker only.

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

The default value is `short`.

Appendix B. Connector Configurator

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

You use Connector Configurator to:

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

Note:

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

The topics covered in this appendix are:

Overview of Connector Configurator

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

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

You use Connector Configurator to:

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

The mode in which you run Connector Configurator, and the configuration file type you use, may differ according to which integration broker you are running. For example, if WMQI is your broker, you run Connector Configurator directly, and not from within System Manager (see "Running Configurator in stand-alone mode" on page 100).

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

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

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

For **connector-specific properties**, however, you need first to define the properties and then set their values. You do this by creating a connector-specific property template for your particular adapter. There may already be a template set up in your system, in which case, you simply use that. If not, follow the steps in "Creating a new template" on page 101 to set up a new one.

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

Starting Connector Configurator

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

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

Running Configurator in stand-alone mode

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

To do so:

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

You may choose to run Connector Configurator independently to generate the file, and then connect to System Manager to save it in a System Manager project (see "Completing a configuration file" on page 105.)

Running Configurator from System Manager

You can run Connector Configurator from System Manager.

To run Connector Configurator:

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

To edit an existing configuration file:

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

Creating a connector-specific property template

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

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

- To create a new template, see "Creating a new template" on page 101.
- To use an existing file, simply modify an existing template and save it under the new name.

Creating a new template

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

To create a template:

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

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

     The names of all currently available templates are displayed in the Template Name display.

- To see the connector-specific property definitions in any template, select that template's name in the **Template Name** display. A list of the property definitions contained in that template will appear in the **Template Preview** display. You can use an existing template whose property definitions are similar to those required by your connector as a starting point for your template.

3. Select a template from the **Template Name** display, enter that template name in the **Find Name** field (or highlight your selection in **Template Name**), and click **Next**.

If you do not see any template that displays the connector-specific properties used by your connector, you will need to create one.

Specifying general characteristics

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

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

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

Specifying values

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

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

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

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

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

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

Setting dependencies

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

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

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

   == (equal to)

   != (not equal to)

   > (greater than)

   < (less than)

   >= (greater than or equal to)

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

Creating a new configuration file

When you create a new configuration file, your first step is to select an integration broker. The broker you select determines the properties that will appear in the configuration file.

To select a broker:

- In the Connector Configurator home menu, click **File>New>Connector Configuration**. The **New Connector** dialog box appears.
- In the I**ntegration Broker** field, select ICS, WebSphere Message Brokers or WAS connectivity.
- Complete the remaining fields in the **New Connector** window, as described later in this chapter.

You can also do this:

- In the System Manager window, right-click on the **Connectors** folder and select **Create New Connector**. Connector Configurator opens and displays the **New Connector** dialog box.

Creating a configuration file from a connector-specific template

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

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

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

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

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

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

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

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

Using an existing file

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

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

- A previous configuration file for the connector.
  Such a file typically has the extension *.cfg.

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

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

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

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

     Choose this option if a repository file was used to configure the connector in an ICS environment. A repository file may include multiple connector definitions, all of which will appear when you open the file.
   - All files (*.*)

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

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

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

Completing a configuration file

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

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

1. Under the **Standard Properties** tab, select the value field for the `BrokerType` property. In the drop-down menu, select the value `ICS`, `WMQI`, or `WAS`.
2. The Standard Properties tab will display the properties associated with the selected broker. You can save the file now or complete the remaining configuration fields, as described in “Specifying supported business object definitions” on page 108..
3. When you have finished your configuration, click **File>Save>To Project** or **File>Save>To File**.

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

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

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

Setting the configuration file properties

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

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

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

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

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

- Associated Maps
- Resources
- Messaging (where applicable)

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

Standard properties differ from connector-specific properties as follows:

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

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

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

Setting standard connector properties

To change the value of a standard property:

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

Setting application-specific configuration properties

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

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

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

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

Encryption for connector properties

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

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

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

Update method

Refer to the descriptions of update methods found in the *Standard configuration properties for connectors* appendix, under "Setting and updating property values" on page 84.

Specifying supported business object definitions

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

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

If ICS is your broker

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

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

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

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

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

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

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

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

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

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

If a WebSphere Message Broker is your broker

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

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

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

If WAS is your broker

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

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

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

Associated maps (ICS only)

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

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

will be used to transform the application-specific business object to the generic business object or the generic business object to the application-specific business object.

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

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

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

- **Business Object Name**

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

- **Associated Maps**

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

- **Explicit**

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

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

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

  To explicitly bind a map:

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

Resources (ICS)

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

Messaging (ICS)

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

Setting trace/log file values

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

To change the logging and tracing values:

1. Click the **Trace/Log Files** tab.
2. For either logging or tracing, you can choose to write messages to one or both of the following:
   - To console (STDOUT):
     Writes logging or tracing messages to the STDOUT display.

     Note: You can only use the STDOUT option from the **Trace/Log Files** tab for connectors running on the Windows platform.
   - To File:
     Writes logging or tracing messages to a file that you specify. To specify the file, click the directory button (ellipsis), navigate to the preferred location, provide a file name, and click **Save**. Logging or tracing message are written to the file and location that you specify.

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

Data handlers

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

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

Saving your configuration file

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

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

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

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

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

Changing a configuration file

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

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

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

- Open the existing configuration file in Connector Configurator.
- Select the **Standard Properties** tab.
- In the **BrokerType** field of the Standard Properties tab, select the value that is appropriate for your broker.
  When you change the current value, the available tabs and field selections on the properties screen will immediately change, to show only those tabs and fields that pertain to the new broker you have selected.

Completing the configuration

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

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

Using Connector Configurator in a globalized environment

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

Connector Configurator supports non-English characters in:

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

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

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

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

Appendix C. HL7 message structure

This appendix describes the HL7 healthcare message structure.

HL7 messages

An HL7 message consists of the following data elements.

Message type

An HL7 message type is a unique identifier for the business purpose of a message. Every message must contain a message type id as way to announce the purpose of the message. For example, ADT is a unique message ID to Patient Administration.

However, it is rather not a unique classification on the structure of a message. One message type can have more than one message structure.

The message type is advertised in the message header segment

Message event

The message event, sometimes called a trigger, is a unique identifier to the context in which message is generated. The message event consists of an upper case letter and two digits. For example, A01 is for admission/visit notification and A61 is for changing consulting doctor. Both A01 and A61 are used with ADT messages.

Event type is advertised in the message header segment.

Message structure

The message structure is a data structure used to express an association of a message type with an event for a class of messages. Each message structure also contains a unique ID.

It structurally consists of a well-defined list of HL7 segments. Segments can be optional, and can repeat. There is no limit on how many times a segment can repeat.

Segments can be aggregated together to form a segment group, which can repeat as well. In the standard specification, segment group is indicated by {} or [], where {} signifies repetition and [] signifies optionality.

Because message structure definition allows {} or [] to enclose single segment, it is possible to interpret the segment group consisting of a single segments. But for purpose of discussing the data handler and related ISBOs, we reserve the term segment group to segment aggregation of two or more segments enclosed by {} or [].

Relative position of segments in a message structure and segment groups is well defined. At the message structure level, segment is the atomic data type.

Message structures are defined by both message type and events. One message type can associate with more than one event, but one event can only associate with exactly one message type. Furthermore, some events with a given message type associate with the same message structure. For example message type ADT with both event A01 and event A04 uses message structure ADT_A01

Table 46. Message structure of ADT_A61

ADT^A61^ADT_A61	**ADT Message**
MSH	Message header
EVN	Event type
[PD1]	Additional demographics
[{ROL}]	Roll
[PV2]	Patient visit--additional information

Segment

Segment is a well-defined list attributes, each of an HL7 data type. All segments start with three upper case letter segment IDs. Segment attributes can be optional and can repeat. The maximum number of how many times an attribute can repeat is specified. Various tables define the validity of certain attribute values.

Note: The relative position of segment attribute is significant in how it is defined.

Every segment ends with a segment terminator, either the ASCII carriage return or x0D.

HL7 calls the segment attribute Field. The standard designates a delimiter, whose value is a defined by the user on per message instances basis, to separate fields.

When the fields repeat, the standard designates a user-defined delimiter, whose value is defined on per message instance basis, to separate repeating fields. Delimiters that separate repeating instances of a field are also defined by use, and call repetition

Table 47. MSH segment specification

SEQ	**LEN**	**DT**	**OPT**	**RP#**	**TBL**	**ITEM**	**Element Name**
1	1	ST	R			00001	Field separator
2	4	ST	R			00002	Encoding character
3	180	HD	O		0361	00003	Sending application
4	180	HD	O		0362	00004	Sending facility
5	180	HD	O		0361	00005	Receiving application
6	180	HD	O		0362	00006	Receiving facility
7	26	TS	R			00007	Date/time of message
8	40	ST	O			00008	Security
9	13	CM	R		0076/0003	00009	Message type
10	20	ST	R			00010	Message control ID
11	3	PT	R			00011	Processing ID
12	60	VID	R		0104	00012	Version ID
13	15	NM	O			00013	Squence number

Table 47. MSH segment specification (continued)

SEQ	LEN	DT	OPT	RP#	TBL	ITEM	Element Name
14	180	ST	O			00014	Continuation pointer
15	2 ID	O			0155	00015	Accept acknowledgement type
16	2	ID	O		0155	00016	Application acknowledgement type
17	3	ID	O		0399	0017	Country code
18	16	ID	O	Y	0211	00018	Character code
19	250	CE	O			00692	Principle language of message
20	20	ID	O		0356	01317	Alternate character set handling
21	10	ID	O	Y	0449	01598	Conformance statement ID

Note: In Table 47 on page 116″Len″ stands for maximum length in number of characters; ″DT″ stands for data type; ″OPT″ stands for optionality; ″RP#″ stands for the maximum number of time the field can repeat; ″TBL#″ stands the id of table, which contains the finite list of legitimate values for validation; ″ITEM#″ stands of the unique id of the data element in the whole standard; ″Element Name″ stands for brief descriptive name of the field.

The structures of many segments are statically defined and can be shared in many message structures. There are also some segments that share the same segment ID but have a different structure -- different attribute in name and data type. QPD, QED, RCP, QAK belong to this category of segments

Data types

HL7 defines a long list of data types. Some are as defined as a primitive type, while others are defined as a complex type, A complex datatype consists of more than one attribute of teh primitive type. HL7 calls the attributes of a complex data type a**component**

For example, the following HD data type uses three components, namespace ID, universal ID, and universal ID type.

```
<namespace ID (IS)>^<universal ID (ST)>^
  <universal ID type )ID)>
```

The standard designates a delimiter, whose value is defined by the user on per message instances basis, to separate components.

Other complex data type can further define a component of a given data type. HL7 calls the attribute of this complex data type **subcomponent**. The complex data type must have all attributes be of primitive type.

The standard also designates a delimiter to separate subcomponents whose value is defined by the user on per message instances basis.

The component delimiter of the data type that defines a component is then demoted to subcomponent delimiter.

HL7 also designates an escape character for escapting characters that are identical to various delimiters. These are defined by the user on per message instance basis.

Most data types are well are statically specified as distinct data structures in the standard. A few data types, notably the CM and * data type, exhibit dynamic behavior, and warrant special attention.

Custom made data types

CM is also called a Composite data type. This data type is a custom construction using previously defined data types for each unique situation, in a different segment, or at different field of the same segment. There are number of data structures associated with the CM data type.

The data structures for the CM types, which can differ widely from one another, are solidified at time when segments they belong to are defined. The CM data type is known at design time.

In earlier versions of the HL7 standard, CM is meant to be custom-made data type, data type defined on local site of deployment. For this reason, CM data type represents endless number of structures with different combination and permutation of readily defined data types.

Version 2.4 of HL7 only has finite number of data structures that is labeled with CM data type.

Polymorphic data type

This is the data type that is marked with * symbol or with ″varies″ data type label, instead of the usual two or three capital letters.

This data type exhibits a common behavior that it can be any one of the already defined data type and the data type is declare in some other place. For example, OBX-2 declares the data type of OBX-5, which is actual data carrier of this polymorphic data type. For ease of discussion, lets call the field that announce the data type of polymorphic data type data Data Type Announcer, and call the field that contains the actual data of the polymorphic data type Data Carrier.

The data type announcer and data carrier can coexist in the same segment; they can also exist on different segments. For example, OBX segment has OBX-2 field being the data type announcer and OBX-5 field being the data carrier. RDF segment contains a field of RCD data type, of which each instance of the RCD field acts as a data announcer for data carrier in the RDT segment. The relationship of the RDF and RDT segment is in analogues to relationship database's table description meta-data and rows of actual data.

In the case of both data type announcer and data carrier residing in the same segment, the relative position of the announcer field and the data carrier field can also vary. Data type announcer can precede data carrier, or the other way around. Section 8.5.2.4, chapter 8, contains example that data type announcer MFE-5 trails behind the data carrier MFE-4.

There is no fixed number of fields that sets data type announcer and data carrier apart. Their relationship is shown in the following diagram.

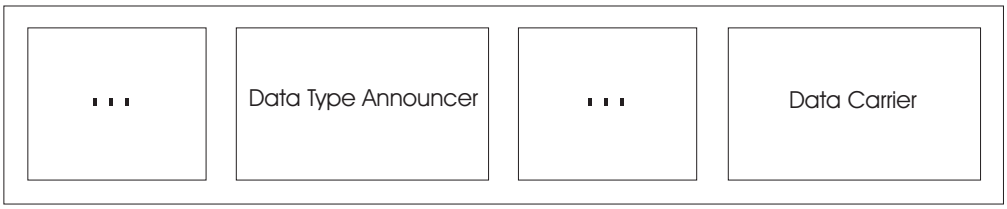

Figure 4. Data carrier and data type announcer relationship in polymorphic data type where "...: represents other fields in the segment.

The range of the variant data type is known and finite, even though it is polymorphic.

Delimiter and enhaced data model

The delimiters for segment, field, component, and subcomponent are all user-defined Delimiters are announced in the message header for each individual message.

A segment delimiter is defined as an ASCII carriage return or x0A. When data containing the characters identical to these delimiters, an escape sequence is used to mark the character.

Component delimiters can be demoted to subcomponent delimiters when a data type is used on anther data type's components. The use of repetition delimiters in most data elements is contextual depending the nature of its data elements, with certain excpetions.

The repetition delimiter is used between repeating instances of data type element segment fields. The common character used is "~". Components of data type element are generally not expected to repeat, except for MA, NA, and QIP data types.

NA is defined as component:

```
<value1> ^ <value2> ^ <value3> ^ <value4> ^ ...
```

The data type of each of its component is NM. It is used to express an array of numbers. The "..." signifies that NA can in theory contain infinitely many values. So it represents a repetition of fields, and the repetition delimiter is the field delimiter. It is illustrated in the following example

```
OBX|3|NA|5&WAV^^99SVL|1|0^1^2^3^4^5^6^7^8^7^6^5^4^3^2^1^0^-1^-2^-3^-4^-5^-
```

Figure 5. The area in bold is an example of how NA is used in the OBX segment.

```
6^-7^-8||||||F|...<cr>
```

MA is defined as component:

```
<sample 1 from channel 1 (NM)> ^ <sample 1 from channel 2
  (NM)> ^ <sample 1 from channel 3 (NM)> ...~<sample 2 from channel 1
   (NM)> ^ <sample 2 from channel 2 (NM)> ^ <sample 2 from channel 3
    (NM)> ...~ ...
```

The pattern in the structure is that a group of clustered components delineated by component delimiter "^" repeats and the repetition is delineated by the segment repetition delimiter "~".

This definition example shows three channels per component cluster, with six channels per component cluster. It is possible to have than more components, depending on the implementation requirements.

Figure 6. The area in bold is an example of how MA is used in the OBX segment

```
OBX|3|NA|5&WAV^^99SVL|1|0^1^2^3^4^5^6^7^8^7^6^5^4^3^2^1^0^-1^-2^-3^-4^-5^-
6^-7^-8~0^1^2^3^4^5^6^7^8^7^6^5^4^3^2^1^0^-1^-2^-3^-4^-5^-6^-7^-
8~0^1^2^3^4^5^6^7^8^7^6^5^4^3^2^1^0^-1^-2^-3^-4^-5^-6^-7^-8||||||F|...<cr>
```

Unlike NA, the size of the component cluster is finite.

QIP is defined as component:`<segment field name (ST) > ^ <value1 (ST) & value2 (ST) & value3 (ST) ...>`

The definition of this data type has two components, the first being a conventional component, and the second having infinitely repeatable subcomponents.

Figure 7. The areas in bold are the repeating subcomponents of QIP

```
|@PID.5.1^EVANS&Param2&Param3|
```

As show, there are three kinds of repetitions in data type elements. NA has repeatable component; QIP has repeatable subcomponents; and MA has repeatable custom defined component clusters. Each of these repetitions uses different delimiters.

Assuming component can repeat with component delimiter, applying the component delimiter demotion rule to the component repetition results in reclassification of these repetitions; the subcomponent repetition can be further considered as in the same category as component repetition.

In order to arrive to a unified data model of HL7 data elements, some expansion has to be made to the traditional HL7 data mode. The enhanced data model allows the repetition of component, where the older data model does not.

The enhanced model permits an addition data element called the component cluster to the data type element. Furthermore both component and component cluster data element are allowed to repeat.

The following diagram depeicts the hierarchy of the data element in this enhanced data model.

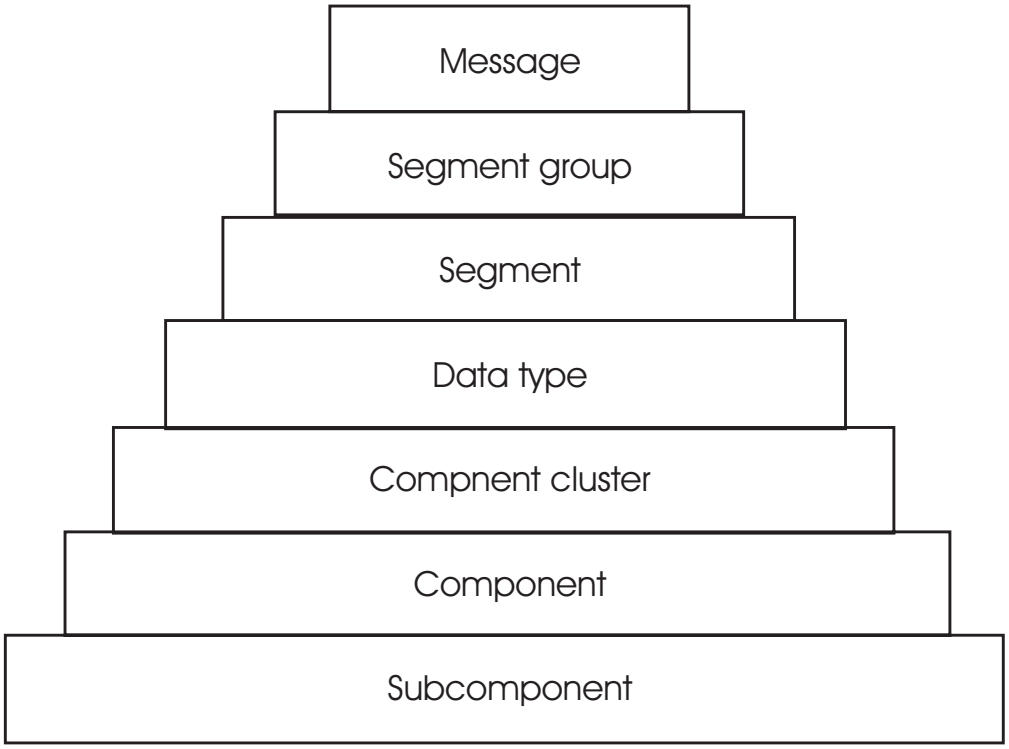

Figure 8. Data element hierarchy in the enhanced data model

The following table summarizes the delimiter for each data element.

Table 48. Delimiters for each data element

Data element in ehanced data model	**Repeatability**	**Delimters**	**Description**
Data type	Yes	Segment field delimiter (commonly ~)	Segment field delimiter delineates repeating instances of the data type element
Component Cluster	Yes	Segment field delimiter (commonly ~)	Segment field delimiter delineates repeating instances of the data type element. Data type, which contains the component cluster, must not be repeatable.
Component	Yes	Component delimiter (commonly ^)	Component delimiter delineates repeating instances of the data type element.
Subcomponent	Yes	Subcomponent delimiter (commonly &)	Subcomponent delimiter delineates repeating instances of the data type element.

Internationalization

HL& allows messages to be encoded using various characters adn encoding, on a per message basis.

The following table details the supported character sets.

Table 49. List of character set identifiers and cannonical names used in the adapter.

HL7 Character Set	Description	Corresponding Canonical Name Supported by BIA_Healthcare Adapter
ASCII	The printable 7-bit ASCII character set. (This is the default if this field is omitted)	(Regular ASCII7)
8859/1	The printable characters from the IDO 8859/1 character set.	ISO8859_1
8859/	The printable characters from the IDO 8859/2 character set.	ISO8859_2
8859/3	The printable characters from the IDO 8859/3 character set.	ISO8859_3
8859/4	The printable characters from the IDO 8859/4 character set.	ISO8859_4
8859/5	The printable characters from the IDO 8859/5 character set.	ISO8859_5
8859/6	The printable characters from the IDO 8859/6 character set.	ISO8859_6
8859/7	The printable characters from the IDO 8859/7 character set.	ISO8859_7
8859/8	The printable characters from the IDO 8859/8 character set.	ISO8859_8
8859/9	The printable characters from the IDO 8859/9 character set.	ISO8859_9
ISO IR14	Code for Information Exchange (one byte)(JIS X 0201-1976). Note that the code contains a space, i.e. ″ISO IR14″.	JIS0201
ISO IR87	Code for the Japanese graphic character set for information exchange (JIS X 0208 1990)	JIS0208
ISO IR159	Code for the supplementary Japanese graphic character set for information exchange (JIS X 0212 1990).	JISO21‘2
UNICODE	The world wide character standard from ISO/IEC 10646-1-1993	UTF-8 (same as IDO1010646
ISO-IR xxxx	Other character sets naming convention layed out in ISO 2375. This is just a different representation of th esame character set id listed ablve.	

HL7 defines the default character set as a single byte character set which should always have ISO IR6 (ISO 646) or ISO IR14 (JIS X 0201 1976) in the G0 area.

Escape sequence for multi-character set data

A message can be encoded using more than one character set and encoding scheme. An escape sequence is used to flag a switch to a different character set, as well as the return to the default character set. The following example shows the data pattern when using escape sequence for data encoded in alternate character set.

```
<Escape sequence>
   <encoded data using alternate character set>
```

Returning from the alternate character set to the default character set uses the same technique using the escape set of the default character set.

```
<Escape sequence>
  <encoded data using alternate character set> <Default escape sequence>
```

ISO 2002-1994 defines the technique on how the escape sequence is structured. by the ISO 2002-1994 standard, the escape sequence is a sequence of bit patterns is used to identify a character set. ISO 2002-1994 uses the decimal xx/yy notation when expressing the escape sequences. This document expresses the bit sequence of escape sequence in a sequence of bytes using hexadecimal notation. The following quote is from the HL7 standard specification, describing where and when to use the escape sequence.

```
Each repetition of a PN, XPN, XON, XCN, or XAD field is assumed to begin with the
default character set.  If another character set is to be used, the HL7 defined
escape sequence used to announce that character set must be at the beginning of the
repetition, and the HL7 defined escape sequence used to start the default character
set must be at the end of the repetition. Note also that several character sets may
be intermixed within a single repetition as long as the repetition ends with a
return to the default character set."
```

Modes of escape sequence for the multi-character set

HL7 provides two modes of specifying and handling escpae sequences:

- ISO 2002-1994 standard
- 2.3 (Described in HL7 v. 2.3

Both modes employ the sam ISO 2002-1994 technique for specifying the escape sequence. They differ in how the escape character is represented in the escape sequence.

ISO 2002 mode escape sequences

This mode uses ASCII escpae characters instead of the HL7 defined escape instead of the previously mentioned characters. The following table provides the list of escape sequences for character sets supported by HL7.

Table 50.

Escape sequence	Character set used in HL7
ESC 2842	ISO-IR6 G0 or ASCII (ISO 646 : ASCII))
ESC 2D41	ISO-IR100 (ISO 8859 : Latin Alphabet 1)
ESC 2D42	ISO-IR101 (ISO 8859 : Latin Alphabet 2)
ESC 2D43	ISO-IR109 (ISO 8859 : Latin Alphabet 3)
ESC 2D44	ISO-IR110 (ISO 8859 : Latin Alphabet 4))
ESC 2D4C	ISO-IR144 or 8859/5 (ISO 8859: Cyrillic))
ESC 2D47	ISO-IR127 or 8859/6 (ISO 8859 : Arabic)
ESC 2D46	ISO-IR126 or 8859/7 (ISO 8859 : Greek)
ESC 2D48	ISO-IR138 or 8859/8 (ISO 8859 : Hebrew)
ESC 2D4D	ISO-IR148 or 8859/9 (ISO 8859 : Latin Alphabet 5)
ESC 284A	ISO-IR14 (JIS X 0201 -1976: Romaji)
ESC 2949	ISO-IR13 (JIS X 0201 : Katakana)

Table 50. (continued)

Escape sequence	Character set used in HL7
ESC 2442	ISO-IR87 (JIS X 0208 : Kanji, hiragana and katakana)
ESC 242844	ISO-IR159 (JIS X 0212 : Supplementary Kanji)

Escape sequence in HL7 define mode

When another character set is to be used, the HL7 defined escape sequence must be used at the beginning of the repetition, and the HL7 defined escape sequence used to start the default character set must be at the end of the repetition.

The escape sequence consists of the escape character followed by an escape code ID of one character, zero, or more data characters, and another occurrence of the escape character where the escape character is defined by the user in the message.

The following table lists the escape sequence for each character set using ″\″ as the HL7 defined escape character.

Table 51. List of escape sequences for HL7-supported character sets

Escape sequence	Character set used in HL7
\C2842\	ISO-IR6 G0 or ASCII (ISO 646 : ASCII))
\C2D41\	ISO-IR100 (ISO 8859 : Latin Alphabet 1)
\C2D42\	ISO-IR101 (ISO 8859 : Latin Alphabet 2)
\C2D43\	ISO-IR109 (ISO 8859 : Latin Alphabet 3)
\C2D44\	ISO-IR110 (ISO 8859 : Latin Alphabet 4))
\C2D4C\	ISO-IR144 or 8859/5 (ISO 8859: Cyrillic))
\C2D47\	ISO-IR127 or 8859/6 (ISO 8859 : Arabic)
\C2D46\	ISO-IR126 or 8859/7 (ISO 8859 : Greek)
\C2D48\	ISO-IR138 or 8859/8 (ISO 8859 : Hebrew)
\C2D4D\	ISO-IR148 or 8859/9 (ISO 8859 : Latin Alphabet 5)
\C284A\	ISO-IR14 (JIS X 0201 -1976: Romaji)
\C2949\	ISO-IR13 (JIS X 0201 : Katakana)
\M2442\	ISO-IR87 (JIS X 0208 : Kanji, hiragana and katakana)
\M242844\	ISO-IR159 (JIS X 0212 : Supplementary Kanji)

Hexadecimal escape sequences and local sequence

HL7 permits the transmission of binary data encoded in the form of `\Xdddd...\`, where the `dddd` denotes s a pair of binary value represented using ASCII character 1-9 and A-F. This is the hexadecimal escape sequence.

Alternatively, up on mutual agreements between parties engaged in the HL7 communication, users can also encode their data using a custom escape sequence, which has the form of `\Zdddd...\`, where the `dddd` are valid characters permitted in TX data type.

Other encoding schemes

Beside character set encoding, there are still other encoding schemes used in the HL7 message standard.

Sometimes, messages make reference to data in other systems. HL7 provides the user the ability to access that data in two ways, by reference and by value. When passing data by reference, it uses the Reference Pointer (RP) data type for the receiving system to track the referenced data, but without physically transferring the data across the wire to the receiving system.

When it is necessary for the receiving system to obtain an actual copy of the data, the data then needs transferred to the receiving system. Many times this data from a third-party application and does not follow the HL7 message construction roles. A special encoding scheme is required to fit for this kind of data into an HL7 message.

There are two ways to encode these special kinds of data: Hex and Base64.

- Hex encoding uses consecutive pairs of hexadecimal digits to represent consecutive single octets of binary data
- Base64 encoding follows the MIME standard RFC 1521. This method uses four consecutive ASCII characters to represent three consecutive octets of binary data by way of direct value to character substitution

The following table is an exact copy of HL7 Table 0290 for the value to ASCII lookup.

Table 52. HL7 table 0290 for the binary to ASCII value in base 64 MIME encoding scheme

Value	Code	Value	Code	Value	Code	Value	Code
3	D	20	U	37	l	54	54 2
4	E	21	V	38	m	55	55 3
5	F	22	W	39	n	56	56 4
6	G	23	X	40	o	57	57 5
7	H	24	Y	41	p	58	58 6
8	I	25	Z	42	q	59	59 7
9	J	26	a	43	r	60	60 8
10	K	27	b	44	s	61	61 9
11	L	28	c	45	t	62	62 +
12	M	29	d	46	u	63	63 /
13	N	30	e	47	v		
14	O	31	f	48	w	(pad)	=
15	P	32	g	49	x		
16	Q	33	h	50	y		

Presentation data

HL7 also allows the exchange of Formatted Text (FT) data intended for the purpose of display rendering. It is often used in the transmission of reports legible to human, rather than machine.

The presentation data contains embedded presentation instructions. Like HTML, there is a group of predefined tags that serve as instruction to the receiving HL7 application on how to render the data. Although the presentation instructions are well defined, the location at where the presentation instructions are embedded in the FT string is completely driven by the design of reports and hence unknown ahead of time.

Message construction rules

Construct the segments in the order defined for the message. Each message is contructed as follows:

1. The first three characters are the segment ID code
2. Each data field in the sequence is inserted in the segment in the following manner:
   a. A field separator is placed in the segment
   b. If the value is not present, no further characters are required
   c. If the value is present, but null, the characters "" (two consecutive double quotation marks) are placed in the field
   d. Otherwise, place the characters of the value in the segment. As many characters can be included as the maximum defined for the data field. It is not necessary, and is undesirable, to pad fields to fixed lengths. Padding to fixed lengths is permitted
   e. If the field definition calls for a field to be broken into components, the following rules are used
      1) If more than one component is included, htey are separated by the component separator
      2) Components that are present but null are represented by the characters ""
      3) Components that are not present are treated by including no characters in the component
      4) Components that are not present at the end of a component need not be represented by component separators. For example, the two data components are equivalent:

         ```
         |ABC^DEF^^| and |ABC^DEF|
         ```

   f. If the component definition calls for a subcomponent to be broken into subcomponents, the following rules are used:
      1) If more than one subcomponent is included, htey are separated by the component separator
      2) Subcomponents that are present but null are represented by the characters ""
      3) Subcomponents that are not present are treated by including no characters in the component
      4) Subcomponents that are not present at the end of a component need not be represented by component separators. For example, the two data components are equivalent:

         ```
         ^XXX&YYY&&^ and ^XXX&YYY^
         ```

   g. If the field definition permits repetition of a field, the repetition separator is used only if more than one occurrence is transmitted. In such a case, the repetition separator is placed between occurrences. If three occurrences are transmitted, two repetition separators are used.)
      In the example below, two occurrences of telephone number are being sent:

      ```
      |234-7120~599-1288B1234
      ```

   Repeat Step 1b while there are any fields present to be sent. If all the data fields remaining in the segment definitino are not present, there are no requirements to include any more delimiters.

The following rules apply to receiving HL7 messages and converting their contents to data values:

1. Ignore segments, fields, components, subcomponents, and extra repetitions of a field that are present but were not expected
2. Treat segments that were expected but are not present as consisting entirely of fields that are not present
3. Treat fields and components that are expected but were not included in a segment as not present

Appendix D. Business object minimal extractor utility

The business object minimal extractor utility is a java-based batch tool that can be used to extract a subset of business objects from a master file of industry-specific business objects. It offers an alternative to using BO Designer. The utility works only with name/value pair text files, and it can be run in any of the following ways:

- To extract individual business object definitions. The utility may be run serially against a master file so that after several runs, a small collection of business object definitions has been extracted. Child objects are extracted each time they are included within a parent object's definition, which means that the same child object definition may be extracted multiple times. This option is invoked by using the -b argument.
- To extract a list of business objects defined in an identified input file. Used in this way, the utility can extract a small collection of business object definitions from one run. If multiple parent objects share the same child objects, the utility will reduce the child object definitions to just one instance, thus reducing redundancy. This option is invoked by using the -f argument.

  If the utilty is run using the -f argument against the output of multiple runs that used the -b argument, multiple copies of a child object definition will be reduced to just one instance.
- To extract a minimal definition of a business object where dependancies of the BO are defined across multiple input files. This option is invoked by using the -c argument.

Example invocation

The following example shows invocation syntax and descriptions of valid parameters.

```
java com.ibm.crossworlds.BusObjMinimalExtractor [-h] [-i <InputBusObjFileName>]
[-b <TargetBusObject>] [-o <BusObjectOutputFileName>] [-d]
[-w <OutputSortedFileName>] [-r <InputSortedFileName>] [-f <BusObjListFile>]
[-c <BusObjDefFileListFile>]
```

Parameter	Description
[-h]	Output the invocation syntax
[-i <InputBusObjFileName>]	Specifies the .in file that contains the business object definition to be extracted
[-b <TargetBusObject>]	Specifies the business object for which you want to extract a definition
[-o <BusObjectOutputFileName>]	Specifies the .in file to which the extracted definition should be written
[-d]	An optional boolean parameter used to indicate that child objects should be listed before their parents. When not used, parent objects are listed first.
[-w <OutputSortedFileName>]	Specifies the name of a topologically sorted file to be created from the <InputBusObjFileName> .in file that contains the business object definitions to be extracted.

Parameter	Description
[-r <InputSortedFileName>]	Specifies the name of a topologically sorted file from which business object definitions will be extracted.
[-f <BusObjListFile>]	A file that contains a list of business objects for which definitions will be extracted. Each business object name should be written on a separate line.
[-c <BusObjDefFileListFile>]	Specifies the name of a file that contains a list of files from which a BO definition will be extracted.

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

Burlingame, CA 94010
U.S.A

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

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

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

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

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

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

Programming interface information

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

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

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

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

Trademarks and service marks

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

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

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

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

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

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

IBM WebSphere Business Integration Adapter Framework V2.4.0

Printed in USA