WebSphere® MQ Integrator Enabler

Application Integration Guide

NOTE:

Before using this information and the product it supports, read the information in *Notices* on page 26.

Fifth Edition (June 2002)

Printed in USA.

Contents

About this book

This publication is intended to illustrate how applications integrate with WebSphere MQ Integrator Enabler (WMQI Enabler). It does not address the construction of adapters other than as a brief discussion of possible approaches to building adapters. In addition, the approach of the document presumes that all applications participating in the enterprise via WMQI Enabler are using XML messages.

Who should read this book

Information technology professionals who will be developing adapters.

Terminology used in this book

All new terms introduced in this book are defined in the *Glossary*.

This book uses the following short names:

- MQSeries: a general term for IBM MQSeries messaging products.
- MQAO: a general term used for the MQSeries Adapter Offering.

Prerequisite and related information

It is assumed that the reader is familiar with XML.

How to get additional information

Visit the following home page at:

http://www.ibm.com/software/mqseries/support/

By following this link you can find:

- The latest information about MQSeries family of products.
- Download Support Packs.
- Access FAQs.
- Access MQSeries family publications library.

How to send your comments

Your feedback is important in helping to provide the most accurate and high-quality information. If you have any comments or suggestions about this book or any other *WebSphere MQ Integrator Enabler* documentation:

- By mail, to this address:

  User Technologies Department (MP095)
  IBM United Kingdom Laboratories
  Hursley Park
  WINCHESTER,
  Hampshire
  SO21 2JN
  United Kingdom

- By fax:

  - From outside the U.K., after your international access code use 44-1962-816151
  - From within the U.K., use 01962-816151

- Electronically, use the appropriate network ID:

  - IBM Mail Exchange: GBIBM2Q9 at IBMMAIL
  - IBMLink: HURSLEY(IDRCF)
  - Internet: idrcf@hursley.ibm.com

Whichever method you use, ensure that you include:

- The publication title and order number
- The topic to which your comment applies
- Your name and address / telephone number / fax number / network ID

Chapter 1
Introduction

This document describe how applications can integrate with WebSphere MQ Integrator Enabler (WMQI Enabler). WMQI Enabler provides the ability to integrate front-end applications with back-end applications in a non-invasive method, using a common XML vocabulary.

The figure below illustrates the intended use of WMQI Enabler:

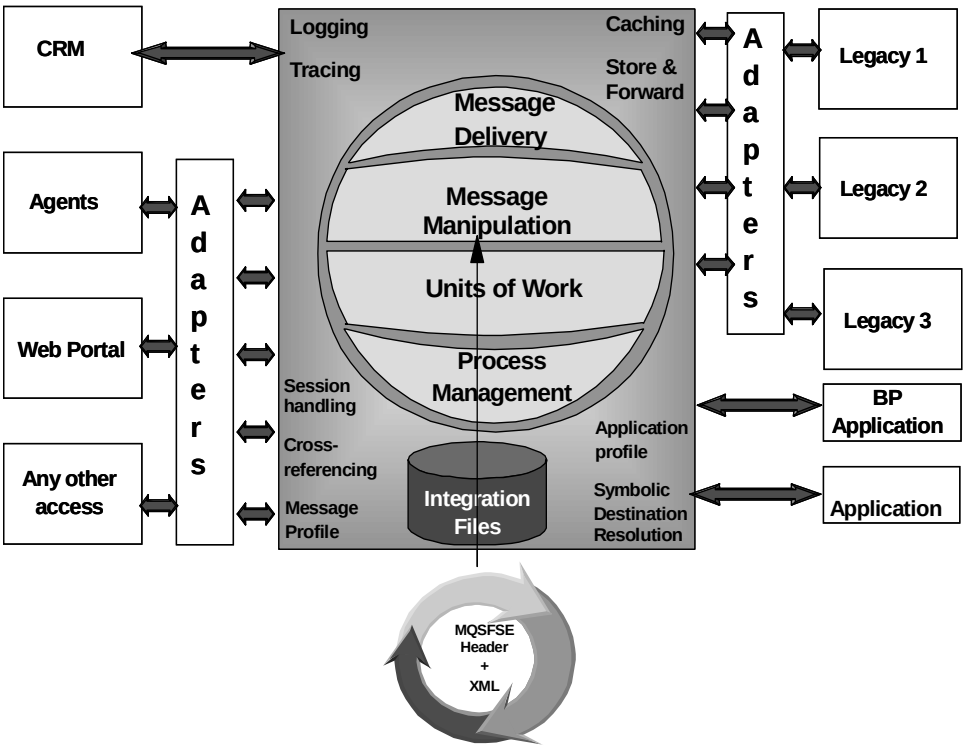

Figure 1: Intended WMQI Enabler use.

The boundary of WMQI Enabler is where application specific information must be translated into the XML message dialect understood by WMQI Enabler. Generally, adapters are used to provide the application to WMQI Enabler semantic translation services, but in some cases, applications contain native support for the required message dialect.

NOTE *Adapters used in this manual have the same meaning as adapter programs.*

Planning for adapter development

The following steps will help ensure that adapters built for WMQI Enabler are sufficiently robust to withstand unanticipated changes to the enterprise:

1. Use cases should be mapped against an appropriate industry specific model in order to validate the proposed messaging objectives. Examples are Insurance Application Architecture (IAA) for insurance, Interactive Financial eXchange (IFX) for retail banking, and Open Application Group (OAG) for generic Customer Relationship Management (CRM) and supply chain activities. The resulting interaction diagrams can then be used to ensure that the anticipated messaging makes sense from a subject matter perspective.
2. Map the data element requirements across all of the WMQI Enabler application participants for a particular use case. This mapping ensures that the messages are sufficiently verbose that all adapters can drive the required processing with the data contained in the messaging.
3. Identify the roles applications will play in the WMQI Enabler integrated enterprise. These roles will add additional requirements that must be handled.
4. Construct the messages.
5. Build the adapters.

The remainder of this document provides the technical details associated with integrating applications for operation with WMQI Enabler.

Adapters are typically used to translate application specific data representations into those supported by the dialect used within WMQI Enabler. Adapters are discussed in this document from the following perspective:

1. The services that an adapter must provide.
2. How messages must be used by applications in order to maintain data integrity.
3. How applications can correlate request and response messages.
4. How adapters drive the WMQI Enabler cross-reference function that allows the adapted applications to process common entities using local key definitions.

Chapter 2
Adapter services

There are two ways to generate WMQI Enabler compatible messages.

The first is to take a message set defined externally to WMQI Enabler and wrapper it with a header that allows WMQI Enabler to process the message. This approach is generally used when some number of applications already support an XML message set and it is decided that this message set is appropriate as the basis for integrating the environment. The "wrappering" process is talked about in more detail later.

The second approach is to use an adapter that translates an application's specific semantics data into an internal message set that was defined specifically as the means for integration using WMQI Enabler. In this case, the adapter does both the message transformation and header creation. Adapters for internal message sets are usually written by the owners of the applications to be adapted, as it requires a great deal of experience with a particular application to map its API's to the appropriate XML message definitions. Building adapters to wrap existing messages on the other hand requires knowledge of the message set.

Regardless of the approach used, messages must be used in a way that maintain the integrity of the applications being adapted. In order to integrate internal message sets with WMQI Enabler, an adapter must provide the set of services described in the following section.

Overview of services

The services that WMQI Enabler compatible adapters must support are as follows:

1. The adapters are responsible for generating the WMQI Enabler header information, which includes:

   Symbolic routing information for defining the application that was the source of the message and the intended target destination of the message (the message profile can also be used to define the symbolic destination).

   Cross reference control information for driving the translation process between system specific keys.

   Message processing information that will be used to drive a specific message though a predefined set of activities.

   Correlation of request and response messages through the WMQI Enabler message correlation infrastructure.

2. The adapters are responsible for loading and unloading the XML message data content by translating application specific data into and out of the appropriate XML representation that includes:

   Creating messages that adhere to the WMQI Enabler XML standard.

   Managing the application data aggregation associated with the XML message content going into and out of the application. Generally, applications do not manage their data at the same granularity that the XML message provides. As a result, the adapter must ensure that the application receives the data in the form in which it is expecting to maintain data integrity.

3. Each interaction between an adapter and its corresponding application must be an atomic unit of work. An application cannot rely on successive messages to maintain consistency; instead, consistent state must be the result of each message interaction. This consistent state is necessary to allow the WMQI Enabler components that are responsible for rollback and recovery, to treat each external interaction as either successfully completed or successfully rolled back to a valid state.
4. The adapter must be capable of informing WMQI Enabler of events that change the required cross-reference key entries for entities managed by WMQI Enabler. All events that cause add, delete, and update actions against managed keys must be communicated to WMQI Enabler. For example, an application adds a party entry; it must send a message to WMQI Enabler that creates a cross- reference entry. That entry can be used later as part of a translation process when the party is referenced in messages to other applications integrated with WMQI Enabler.
5. Adapters must define their response queues using the MQSeries Message Descriptor (MQMD) infrastructure.
6. Adapters must provide support for recovery of a message if the adapter's application should encounter problems such as a failure. Although WMQI Enabler has full recovery capability, WMQI Enabler may require the adapter to recover the message processing.
7. The adapters are responsible for handling application security requirements. The adapters use the information in the user identification tags in the WMQI Enabler and MQMD headers to satisfy application authentication and authorization requirements. In some cases, this determination requires custom application changes.
8. Logging is utilized for the purposes of creating audit trails for activities in and out of the application that the adapter is driving.

The way these services are implemented is based on loading the appropriate information into WMQI Enabler messages.

The WMQI Enabler message header is illustrated in the figure below:

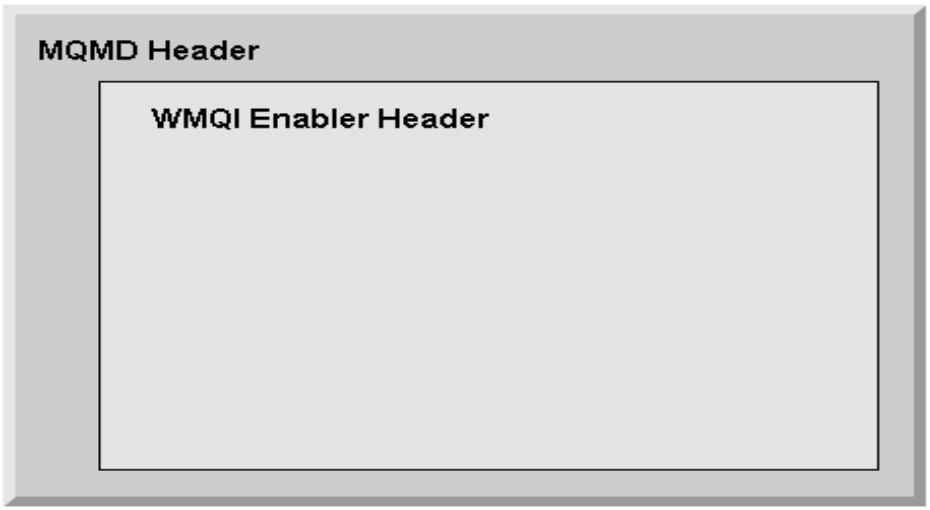

Figure 2: WMQI Enabler message header.

The MQSeries Message Descriptor (MQMD) is the standard MQSeries message header and is used extensively by WMQI Enabler components to drive processing through the MQSeries Family of products. The WMQI Enabler header is used to specifically drive processing through the WMQI Enabler functions. The remainder of the message is the data content represented by the selected XML language dialect.

MQMD header

WMQI Enabler uses the MQMD header to drive MQSeries processing. This means the adapter must load appropriate values within the MQMD. Conventions must be established for the enterprise that define the kinds of values all adapters must use.

An example is the Report and Feedback fields within the MQMD. These fields drive processing that is not specifically related to WMQI Enabler. They are instead related to how MQSeries itself provides feedback to applications about messaging activity. It is important that all WMQI Enabler implementations spend the time defining what features and services will be used from the MQSeries Family of products and explicitly determining what is related to WMQI Enabler processing. For more information on MQMD, fields see the ***MQSeries Application Programming Reference***.

Certain MQMD fields are directly manipulated by WMQI Enabler. These fields are:

Expiry: This is a period of time expressed in tenths of a second, set by the application that puts the message on the queue. The message becomes eligible to be discarded from the destination queue once this period of time elapses.

MsgId: This is a byte string that is used to distinguish one message from another.

CorrelId: This is a byte string the application can use to correlate one message to another or to other work that the application is performing.

ReplyToQ: This is the name of the message queue to which the receiving application should send it's reply messages (MQMT_REPLY) and or report messages (MQMT_REPORT).

ReplyToQMgr: This is the name of the queue manager to which the reply message or report message should be sent.

UserIdentifier: This is part of the identity context of the message. It identifies the user who originated the message to the MQSeries Family infrastructure, and is used to validate access to the various MQSeries Family components used in the WMQI Enabler.

MsgType: This indicates the type of message being processed. WMQI Enabler requires that request messages use the MQMT-REQUEST flag and response messages use the MQMT-REPLY flag.

The way in which WMQI Enabler uses these fields is discussed in more detail through the remainder of the document. WMQI Enabler also uses fields in the WMQI Enabler Message header to drive processing. The WMQI Enabler header fields are defined in the following section.

WMQI Enabler message header

Once the MQMD fields have been filled in with appropriate values, the WMQI Enabler message header fields need to be loaded based on the content and context of the XML command an adapter wishes to execute. It is important to address the specific set of fields that are used to drive processing.

The following list of fields are associated with integrating applications with WMQI Enabler. The message header field names and their usage is as follows:

sourceLogicalId: This is the symbolic name used to represent the application that generated the message. This value can be resolved by the Symbolic Destination Resolution (SDR) function into a specific queue and queue manager name for routing purposes. Additionally, this value can be used within MQSeries Workflow to identify the process associated with the message request. This value should also match the value coded in the AlternateId tag attribute named sourceLogicalId.

destinationLogicalId: This is the symbolic name used to represent the application to which this message is to be routed. This value can be resolved by the SDR function into a specific queue and queue manager name for routing purposes. Although the WMQI Enabler message header supports multiple **destinationLogicalIds**, one for each CrfActionGroup, WMQI Enabler currently only routes based on the first **destinationLogicalId** encountered.

authenticationId: This ID is to be used for authorizing access to specific functions. This value can be used within MQSeries Workflow to identify and validate the role of the entity requesting this message. This value may be different from the UserIdentifier in the MQMD which is used to validate access to MQSeries family resources. WMQI Enabler also uses the authenticationId as a part of the simple authorization process provided with a WMQI Enabler Logon command.

sessionId: This is the session ID supplied by WMQI Enabler when a Logon command is processed. It is used during WMQI Enabler session validation.

bodyType: This field is used to identify the type of message to be processed. For example: WorkFlow indicates that this message was generated by WorkFlow, HUBONLYONLINE indicates this is a hub only online message, and IAA-XML indicates this is a message with an IAA structure.

publish: This field indicates whether or not publishing is desired for this message. This field can be overridden by the message profile.

ErrorInfo: This structure is intended to contain error information associated with WMQI Enabler processing, and is used by WMQI Enabler itself.

bodyCategory: Body category is used to identify the specific MQSeries Workflow process that is used for this message type. This tag is also used to identify the message type to WMQI Enabler. It can, but is not required to be the same as the command value that it is connected to by the CrfActionGroup reference ID. The contents of this tag no longer requires the request/response suffix as it did in the previous version of WMQI Enabler. Instead, request/response is indicated by the MQMD MsgType.

KeyGroup: KeyGoups are defined per aggregate contained in a particular command. Related by refId. If an aggregate has multiple keys, multiple alternate ids are needed within the KeyGroup. This assumes all keys are related by UUID and have the same keyGroupType. There can be multiple KeyGroups per CrfActionGroup depending on the structure of the XML command.

NOTE: *There can be more KeyGroups than there are aggregates for a command, but there has to be at least one KeyGroup for each aggregate containing a reference Id. The reason for this is that the number of KeyGroups*

has to be flexible enough to represent the appropriate activities for the CRF in order to maintain the required UUID relationships, but each aggregate requires a key so that it can be referenced.

The fields' destinationLogicalID and bodyCategory can be loaded as default values and then overridden for each command. This allows the potential for each command to be handled independently. However, this feature is not currently supported by WMQI Enabler and would require customization to enable.

Now that the fields that control processing have been identified, it is necessary to describe how applications and their associated adapters must use the XML messaging semantic.

Below is a sample WMQI Enabler header:

```
<Message id="M5441920" sessionId="2914320" version="1.4"
bodyType="XML" timeStampCreated="2000-10-22-08.00.00"
sourceLogicalId="FrontEnd" destinationLogicalId="BackEnd"
authenticationId="SysAdmin" crfCmdMode="alwaysRespond"
publish="true">
<Default>
    <DefaultCurrency>USD</DefaultCurrency>
</Default>
<CrfActionGroup bodyCategory="AddPerson" crfPublish="true"
     crfCmdMode="alwaysRespond" destinationLogicalId="BackEnd">
     <CommandReference refid="CMD1"/>
<KeyGroup id="K1" keyGroupType="Person">
     <AlternateId value="123450050" sourceLogicalId="FrontEnd"
state="add"/>
  </KeyGroup>
<KeyGroup id="K2" keyGroupType="PostalAddress">
     <AlternateId value="123450050" sourceLogicalId="FrontEnd"
state="add"/>
  </KeyGroup>
<KeyGroup id="K3" keyGroupType="TelephoneNumber">
     <AlternateId value="123450050" sourceLogicalId="FrontEnd"
state="add"/>
    </KeyGroup>
  </CrfActionGroup>
 <ErrorInfo>
<COMMAND>
```

Wrappering

As stated earlier, wrappering is the process used to add an WMQI Enabler header to a message so that it can be used by the hub.

Wrappering is both a flexible and an extensible way to provide the capability of handling a variety of business situations.

The wrappering message structure is illustrated in the figure below:

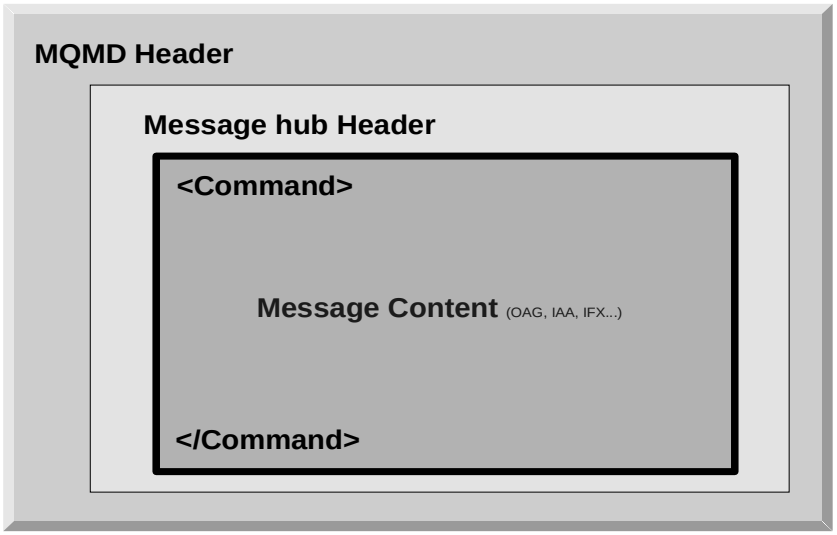

Figure 3: WMQI Enabler message structure for IAA.

Specific examples of the XML architecture are offered in Industry Reference Manuals to assist in the preparation of the adapter programs. The WMQI Enabler header contains the critical information for the product including the identification of the XML architecture being employed. Therefore, the product users, while restricted to XML, are not confined to one XML dialect, but may decide what architecture fits with their strategic objectives.

Implementing Wrappering Adapters

A key issue that must be addressed when wrappering a message set is to determine the topology. Regardless of the approach, care must be taken to insure that the capabilities of the adapters are addressed as a part of an WMQI Enabler implementation. In the case of the MQAO example there are essentially two approaches.

First, adapters can be written to generically deal with a specific message.

Second, adapters can be written to support specific applications.

The approach taken depends on the type and number of applications to be integrated. Generally this ends up being a decision on whether to duplicately deploy message oriented adapters (adapting systems), or to externally implement routing (adapting specific messages). When adapting for systems,WMQI Enabler provides the symbolic destination capabilities. The challenge for the adapter writer is to determine how to create the proper combination of adapters to support the specific message requirements of an application. Essentially some number of "fat" adapters or physical grouping of adapters is created for an application. The hub delivers the messages to the application's queue and the adapter group processes the messages for the application.

The other option is to route all messages to a particular type of adapter designed for those types of messages. In this case WMQI Enabler is merely resolving the symbolic destination to a specific queue of an adapter, and the adapter technology itself must then determine where to deliver the specific messages.

Regardless of the approach the adapters must still insure that the messages are correlated properly and returned to the replyToQueue and replyToQueueManager as specified by WMQI Enabler.

Chapter 3
Requirements for generating messages

In order to properly integrate applications with WMQI Enabler the appropriate message set must be selected. WMQI Enabler supports external standards based messages like OAG, IAA, and IFX. WMQI Enabler also supports internal message sets based off of the WMQI Enabler interface design model (IDM).

Once the message set is selected, there are a number of requirements that must be satisfied when generating messages that can be successfully processed by WMQI Enabler.

1. Messages must be well-formed and valid XML messages based on the XML message specification and the appropriate DTD. This ensures that the messages will be usable through out the WMQI Enabler integrated enterprise. While it is certainly possible to write custom flows within WMQI Enabler to validate message content, by default WMQI Enabler relies on the end-points to do this validation (although it is possible to update message content as a part of processing within a workflow using MQSeries Workflow). When the transactional message processing supervisor is used, the activities can request access to and update any tag within the XML message. Access to the message content is made available within the MQSeries Workflow activity and the MQSeries Workflow profile table.
2. End-points must ensure that the generated XML messages maintain application integrity as these applications transition from one valid state to another. WMQI Enabler architecture presumes all workflow activities are supported by message implementations that can be processed as complete units of work in order to facilitate restart and recovery. As long as the application is in a valid state, after completion of a message, then this requirement has been satisfied. An example, in the case of a fictitious GetAgreement message that contains both policy and client information would be to store the client information with a pending status that would not be activated until the GetAgreement response message is received.

There are two reasons for an application to generate messages.

1. To respond to a message.
2. To notify the rest of the systems in an WMQI Enabler infrastructure when data has changed.

Application roles

As stated earlier, the role an application plays dictates specific requirements for the Implementation of the use case within the environment. The adapter for an application generally does not need to know the role the adapted application is playing, unless certain cross referencing design patterns are used. See the ***Adapters and Cross-Referencing*** section "***Approach for using CRF option 1 (CRF without rollback)***."

System of truth

When the role of the application is the system of truth, then there are special requirements. “System of truth” refers to the singular system in the enterprise responsible for maintaining the correct information for an entity. There may be multiple systems of truth for a type of entity, such as clients, but there should be a clear delineation as to which system manages which client. If there is not, then the *systems* that redundantly make up the *systems of truth* must be treated as one entity for WMQI Enabler. In this case, a workflow is required that interrogates the success of the *systems of truth* to insure that events are processed such that the applications remain synchronized.

Regardless of who the system or systems of truth are, all requests for message processing should be sent to it in order to preserve data integrity throughout the topology. Routing requests to the system of truth allows those systems to insure that the event is validated by applicable business rules. This routing is generally handled within WMQI Enabler by implementing an appropriate MQSWF workflow to ensure message requests are sent to the appropriate target applications. See the WMQI Enabler ***Planning Guide*** and ***Development Guide*** for more details on how these tasks are accomplished.

It is the responsibility of all integrated applications to publish the result of a message event to the hub so that it can be received by other interested parties in the environment. In some cases, this publishing may be accomplished via a simple response message, other times it may require the use of a publish and subscribe mechanism. The decision is made based on the requirements captured in the use case and the steps documented earlier.

Non-system of truth

When the application is not the system of truth, the adapter is responsible for notifying the system of truth of the changes to internally managed data; as well as requests for changes to data managed externally to the application. This type of notification means applications must be aware of their internal activities to trigger the appropriate XML messages to be generated. In this case, MQSWF is used with WMQI Enabler to handle message delivery to the appropriate applications.

The application should also be capable of handling messages when the system of truth notifies that the request has been rejected. In this case, the originating system now needs to adjust its data appropriately, which is defined based on how the use case specifies to handle this situation.

Responsibilities when using publish/subscribe

Two message delivery methods were described. The first was MQSWF driven. The second was MQSeries Integrator Publish/Subscribe.

NOTE *Access to Publish/Subscribe could be MQSWF driven, but the difference between this and using MQSWF activities alone is the indeterminate nature of when a subscription is satisfied by a publication. As a result, when Publish/Subscribe is used the originating application must be prepared for the latency inherent in this approach.*

The latter approach adds a few additional requirements on an adapter. When MQSeries Publish/Subscribe is used, the adapter is responsible for insuring the sequencing of internally processed events, with those coming from the subscription queue to ensure that processing occurs in the correct order.

An additional requirement is that by convention, adapters generating publications and those using subscriptions must be kept in sync by defining topics at the level of granularity matching what is specified in the XML vocabulary's aggregate definitions. The reason for this is the aggregates specify the granularity of the keys within the CRF, thus defining the level of detail at which applications can communicate through WMQI Enabler. This complicates matters because as previously discussed, XML aggregates may not match internal application data requirements. See the ***MB-XML Architecture Book*** for more insurance industry specific details.

MQSeries Publish/Subscribe functionality can assure delivery of published data, but only the current version of the publication is retained. Using current functionality, published messages must be sufficiently verbose allowing subscribers access to whatever message content is required to synchronize the environment. In the future it may be possible to publish only the changed data, currently this would require customization to the WMQI Enabler environment.

Another requirement is that system of truth applications must be able to version the data they own so that they can determine if updates are based on the proper revision of an aggregate. It is assumed that only the systems of truth will publish updates, since they own data integrity for the information being published.

Change CRF information

In many cases it is necessary to process changes that affect the keys contained in the cross reference file. WMQI Enabler provides two mechanisms to support changes to CRF information.

The first is to imbed the proper key state information within messages flowed through the hub. The CRF will analyze the keygroups containing the CRF entries in the form of alternateId entries and make the appropriate adjustments. Normally this processes is done as a part of the response message processing and is sufficient when a deterministic request response model is used.

The second is similar to using publish and subscribe, there is no explicit requirement for a response message. An example is the receipt of a message via a subscription queue, where the contents of the message dictate CRF changes for locally managed data. In this case, an application must use a HUB_Only CRF message. The HUB_Only CRF messages allow the application to send explicit commands to the WMQI Enabler CRF function. An example would be to register a key for a new party aggregate that is received via a subscription to an AddParty topic and persisted locally. The HUB_Only CRF command must be used by the application to keep the CRF file synchronized with the enterprise.

The following is a set of rules governing CRF activity:

- If a CrfActionGroup does not have a destinationLogicalId attribute when entering the CRF subflow, if the destinationLogicalId on the message tag exists it will be copied to that CrfActionGroup's destinationLogicalId attribute.The message will leave the CRF subflow with this change.
- If no destinationLogicalId attribute is found on the message tag then their must be one on every KeyGroup that contains an AlternateId with the state of exists.
- There can be only one AlternateId with the state of exists per KeyGroup and it must be the first AlternateId in that KeyGroup.
- All AlternateId(s) with the same state will be processed together in the order in which they appear in the message. These groups of AlternateId(s) with the same state are processed in the order exists, delete, modify, and add.
- The state attribute of an AlternateId is converted to all lowercase letters before matching is done. When a state is updated, after the operation is complete, it is in all lowercase letters.
- Validation of all CrfActionGroup(s), KeyGroup(s), and AlternateId(s) is performed before any processing is started.

- All AlternateId(s) within a KeyGroup that are present in the CRF database must be connected to the same UUID. If any AlternateId(s) with the states of exists, delete, and modify are found connected to a different UUID than is either specified in the KeyGroup or is found connected to other AlternateId(s) in that KeyGroup of the same three state, then a error is declared and an exception is thrown.

- All AlternateId(s) with the states referenced, deleted, modified, and added are ignored by the CRF completely. No validation or processing is done on these AlternateId(s).

- An AlternateId or one set of value, sourceLogicalId, and keyGroupType can be connected to one and only one UUID.

- The **attributeString** held for an AlternateId in the CRF can be modified. To modify an **attributeString**, specifically the attributes **value** and **newValue** to hold the same string.

- KeyGroup attribute tag **uuidLookup** is processed as follows:

  If uuidLookup="true" **AND** the KeyGroup includes a UUID element **AND** the KeyGroup does not include any AlternateId elements, then the UUID value supplied is used for the CRF lookup. The CRF processsing done using that UUID is identical to the processing performed when an AlternateId with state="exists" is provided.

  If the uuidLookup="true" **AND** the KeyGroup includes an AlternateId element with a state="exists", then the UUID value is supplied even if the destinationLogicalId does not have an entry attached to that KeyGroup.

  In all other cases, the state or absence of uuidLookup is ignored.

Chapter 4
Adapter message flows

Processing a message

As previously discussed, adapters drive messages through WMQI Enabler, and process messages from WMQI Enabler based on specific fields in the MQMD and WMQI Enabler message headers. Essentially the source application adapter puts a message in the HUB_IN queue for WMQI Enabler and sets the appropriate fields causing WMQI Enabler to manipulate and eventually deliver the message to the appropriate target applications. Assuming a MQSeries Workflow is required, WMQI Enabler processes messages by attaching the message to the appropriate workflow script. The adapters are responsible for ensuring the proper workflow process is activated based on the values loaded into the MQMD and WMQI Enabler message headers.

The figure below illustrates this process:

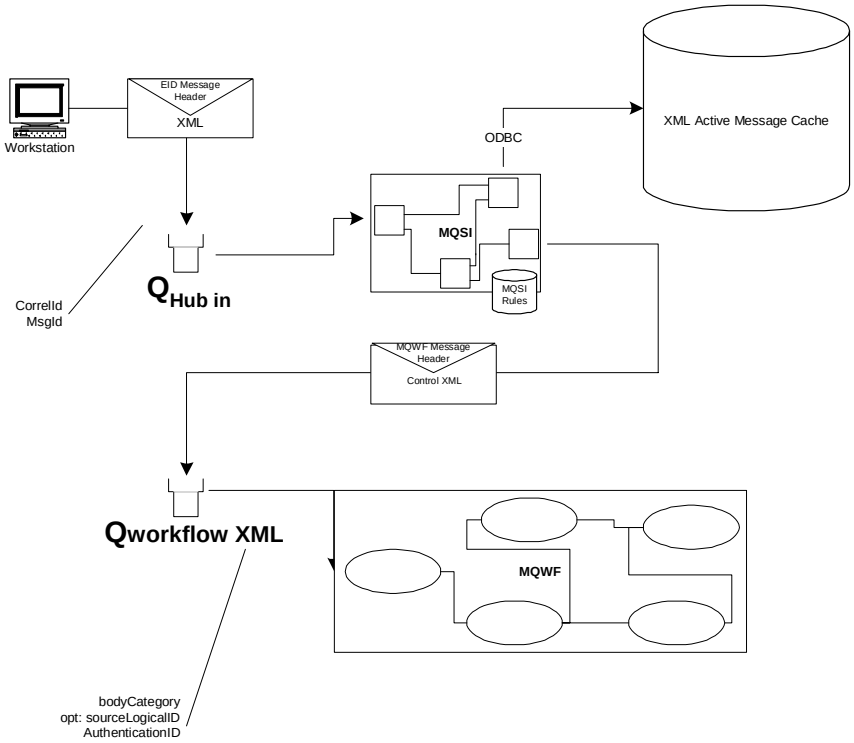

Figure 4: Processing a message to the appropriate workflow.

Sending a request message

The source application adapter must indicate that the message is a request by setting the MQMD msgType field to MQMT_REQUEST. The ReplyQ and ReplyQMgr must be set to indicate where a response should be delivered. The CorrelId and MsgId are set using the default MQMD value of copy MsgId to CorrelId so the request can be correlated. The key to processing the message based on the use case requirements is that the value of the bodyCategory tag contains the name of the workflow process to execute. The Workflow process can also be designated within the message profile of the message type indicated in the bodyCategory. MQSeries Workflow is designed to execute the requirements of the use case as described in the message usage section. Once the message is in MQSeries Workflow, then various functions may need to be called to manipulate the message.

The figure below illustrates this process:

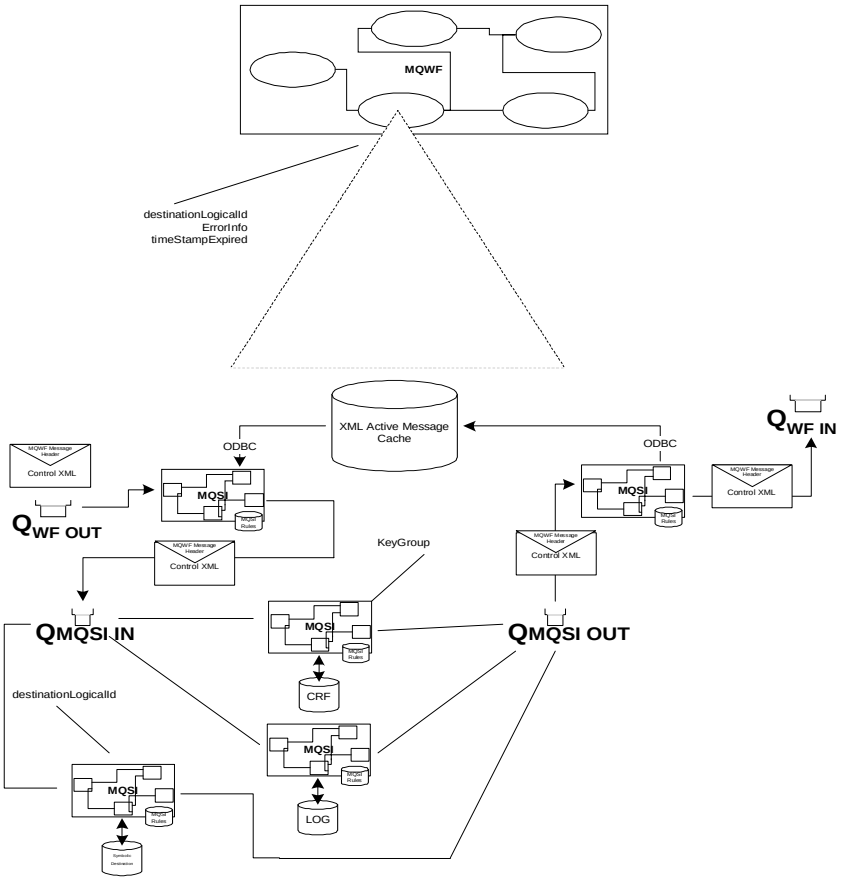

Figure 5: Workflow processing of message.

In the case of lightweight message processing, the message is delivered directly to the target adapter without invoking a workflow.

Ensuring delivery to target destination

The MQSI functions of WMQI Enabler use the destinationLogicalId as a way to find the queue name associated with the target application for a message. Both the adapter and the workflow template created using MQSeries Workflow can set the value of this field. The cross-reference function uses the values coded in the KeyGroup to do key translations. The cross-reference will be addressed in more detail in the following section. The workflow template can update the message content information also update the message timeStampExpired information. The final aspect of driving processing from the adapter point of view is accessing a target application and adapter.

The following figure illustrates this process:

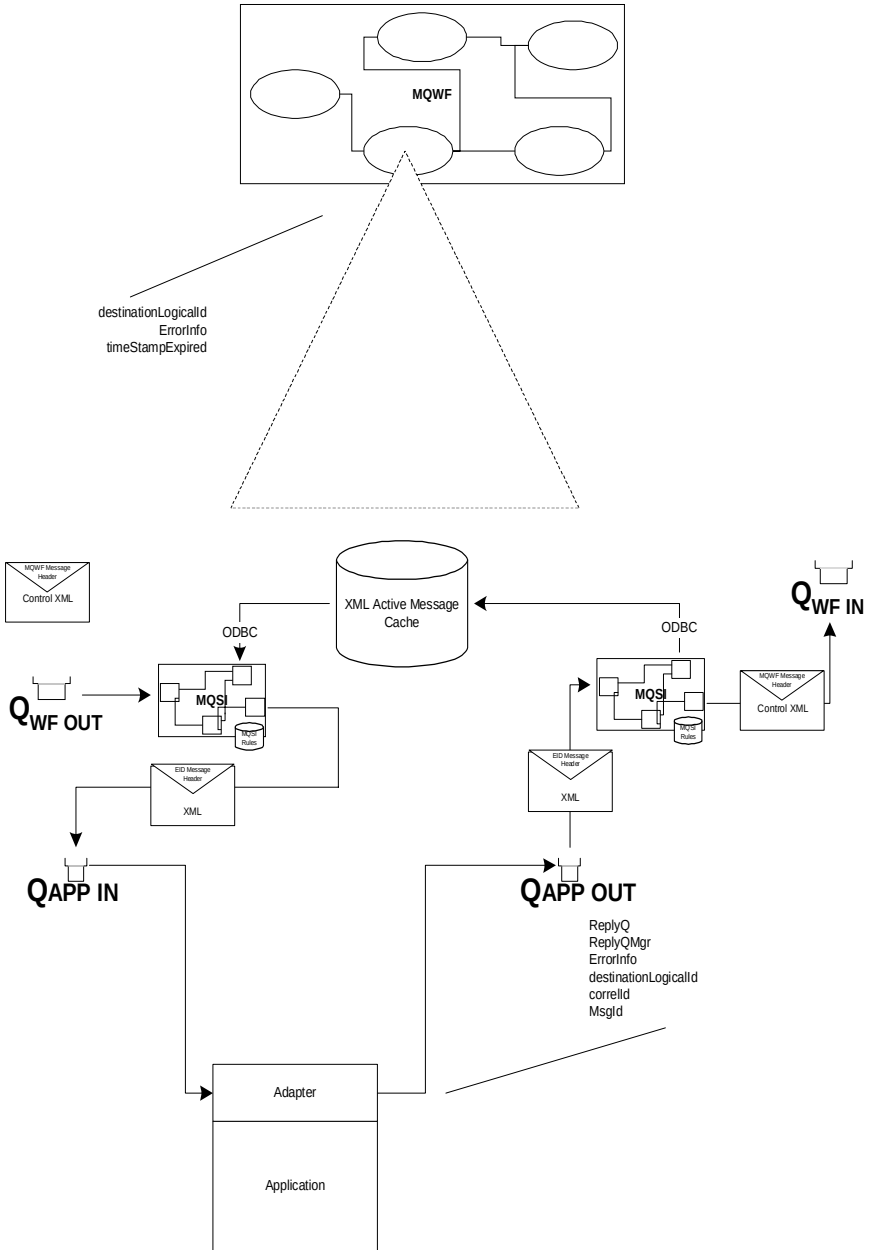

Figure 6: Final processing of message.

Reply

At the simplest level, the target application adapter must look into the XML command and determine what message-processing mode is being requested. While there are several modes, the one that requires specific actions associated with WMQI Enabler is request/reply. In this case, the target adapter must indicate that the message is a response by setting the MQMD msgType field to MQMT_REPLY. The MsgId and CorrelId must be set based on the option set in the MQMD Report field. There are many approaches to correlating messages, but WMQI Enabler has been specifically designed to support the Correlation-identifier option in the MQMD, MQRO_COPY_MSG_ID_TO_CORREL_ID, which indicates copying the message identifier to the correlation identifier. This is the default value and an approach supported by MQSeries. The way the response is delivered to the source adapter and application is via the values coded in the MQMD ReplyQ and ReplyQMgr.

At this point, the creation of the appropriate message and the routing of that message through WMQI Enabler has been handled. The remaining issue is to handle the cross-referencing of application specific keys so that applications can reference the same entities using their native keys.

Chapter 5
Adapters and cross-referencing

Cross-referencing is an optional feature for WMQI Enabler, that provides database key translations between applications when using XML message models that expose low level database keys. When used, end points not only have to define how the message should process through WMQI Enabler, via the settings loaded in the MQMD and the WMQI Enabler message header, but they are also required to control how key cross-referencing should work between applications. The reason for this is that key cross-referencing requirements closely mirror the use cases that the message processing scenarios are designed to support. Key cross-referencing designs should take into account, unit of work requirements in order to ensure the cross-reference file stays synchronized with the applications being supported. As a result, the best-practices rules for handling exception conditions when processing applications must also be considered.

The Cross Reference Function (CRF) supports the following based on the value loaded into the state tag, which is an AlternateId attribute and is used to describe the state of the key. The CRF changes the value of the state attribute once the AlternateId has been processed. The definitions are as follows:

Initial state	Definition of processing	Final state
add without UUID (Create)	This new UUID entry including the values coded in the AlternateId tag.	added
add with UUID (Attach)	Since the UUID exists, this is a request to attach a new AlternateId entry to an existing UUID.	added
referenced	Indicates that the key should be ignored.	referenced
exists	Indicates that the key should be translated to the value held for the system *coded in the sourceLogicalId* attribute field of the AlternateId tag.	referenced

Initial state (Continued)	Definition of processing	Final state
modify	Indicates that the value of the AlternateId attribute *value* should be replaced by the value of the AlternateId attribute *newValue*.	modified
delete	Indicates that the value of the AlternateId attribute value should be invalidated from the UUID.	deleted

Table 1: CRF supported functions.

The CRF cycles through all AlternateIds within a KeyGroup, all KeyGroups within a CrfActionGroup, and all CrfActionGroups within an WMQI Enabler message header. The responsibility of adapters is to setup the KeyGroups in such a way that the cross-reference function maintains and translates keys as required to support the WMQI Enabler integrated applications. This KeyGroup setup means for each state of a key used as a part of message processing (add, modify, exist, and delete), the adapters must drive the Cross Reference Function consistently for each aggregate contained in the message. This setup is important because aggregates must have state codes that will cause the Cross Reference Function to do things with the keys for the aggregate that facilitate the requirements laid out in the use case.

As a result, a message may indicate a Claim is to be added. There will be a Claim aggregate with a state of add, but their may be other aggregates in the message that may contain Policy information. In this case, these aggregates were not added and the state tag should be set to the value reflected in the use case. The way that the state tag values are determined is based on the use cases that define the requirements for how to manage the information passed through the enterprise and the conventions used to satisfy the cross-reference processing requirements. The next sections illustrate examples of how the CRF can be used with IAA-XML messages. Regardless of the selected message dialect, the processing described remains the same.

Approach for using CRF, option 1 (CRF without rollback)

As an example, a front-end application sends an addParty message to the party system. For this to occur, the front-end system creates a keygroup representing each of the aggregates associated with the IAA-XML AddParty message command. The keygroups contains the front-end system's keys for each of those aggregates. At this point, the state should be set to “reference” so that the CRF ignores the keys.

NOTE: *This would be appropriate if the front-end is not the system of truth for party and there is no need to use MQSeries Workflow to handle potential cross-reference table back out scenarios.*

The party system receives the message from WMQI Enabler and processes the addParty message. The party system adapter writes an IAA-XML AddParty response message to the front-end indicating the success or failure of the process.

When the add is successful, the party system puts the key information for the front-end and back-end systems in the AddParty response message. (Again, this is being done to simplify the rollback requirements for the CRF). This addition includes keys for all of the aggregates in the request and response messages. The reason all of the aggregates must be processed is because relationships must be setup in the CRF for all of the aggregates contained within the AddParty message.

In order to do this key information addition, all of the key relationships must be added to the cross-reference tables at the granularity that XML specifies. This statement means there is not a one-to-one relationship between the keygroups and the aggregates in the command. If the command contains aggregates with keys (implemented as reference ids), there will be at least one keyGroup for the aggregates. This use of reference ids ensures the CRF has access to all of the KeyGroups required to maintain the necessary relationships and ensures that each processed aggregate is represented by a UUID.

Using the CRF at the aggregate level of granularity ensures messages will not have to be remodeled in the future as business requirements and system participants change. In order to use the CRF at this level, the state information for the keys should be set to “add” within the response message. CRF will create a new UUID for the first AlternateId contained in the response message and then store the UUID in the message. The CRF will then see the UUID, and the remainder of the AlternateIds will be treated as attachments for the remaining keys in the message. This is the approach illustrated by the sample WMQI Enabler use case templates in the Model Office.

Alternate approach for CRF, option 2 (CRF with rollback)

It is important to remember that WMQI Enabler does not dictate how flows using CRF are implemented, only that the flows maintain the integrity of the environment. Another approach would be to have the front-end system send the request message with the state tags for all of the keys set to “add”. This approach will cause the CRF to create a UUID and attach the keys to the UUID as new entries in the CRF. The message would then be sent to the target application with the state tag updated to “referenced”, and the new UUID included.

In this case, the target application will generate a response to the original request. This response will be routed back to the front-end system that originated the request. If the processing was successful, the back-end loads its own keys for the added information with state tags set to “add”. The alternate ids from the original request can be cleared or ignored since the CRF set their state tags to “referenced”. However, the UUID must be copied from the request message to the response message so the CRF will see it and treat the “add” states as "attaches" for the back-end system's keys in the response message.

Again, there will be keyGroups for aggregates that are not contained in the response message, because we are building the relationship between the front-end and bank-end systems for the content of the add message. There will also be keyGroups for any aggregates in the response message itself so the CRF can do the appropriate cross-referencing as the response is delivered back to the originating application. If the processing failed, the back-end does not have to load its keys in the response. However, the front-end seeing the add failed, would be responsible for deleting the keys it had originally asked the CRF to add in the initial request.

Both approaches are valid. WMQI Enabler provides the flexibility to use either for the systems being integrated, based on the requirements detailed in the use cases. The only requirement is that whichever approach is being used, it must be used consistently. The CRF must be kept in sync with the business content of the messages being processed through WMQI Enabler.

CRF processing requirements

The following is a list of processing requirements based on the cross-reference functionality provided in WMQI Enabler.

1. WMQI Enabler supports either including a UUID or using the AlternateId tag and associated attributes to indicate a CRF entry.
2. If a KeyGroup has an AlternateId with the state of “exist”, then it must be the first AlternateId in the list of AlternateIds for a KeyGroup.

3. Adding new keys in the form of AlternateIds to an existing UUID requires either an AlternateId with the state of “exists”, so the CRF can first determine the correct UUID and where to attach the remaining AlternateIds or a UUID.
4. Adapters must use the sourceLogicalId attribute of the AlternateId tag in order to find the correct AlternateIds. The order of AlternateIds within a KeyGroup, or even the KeyGroups themselves is not guaranteed.
5. The CRF does not remove AlternateIds from key groups. Response messages can either start new KeyGroup AlternateId lists or include the entries from the original request messages. This technique allows the CRF to change the state of the AlternateId entries as they are processed.
6. Keys must be processed based on all of the aggregates defined in the messages. The cross-reference function is not designed to maintain relationships between KeyGroupTypes. Instead, the KeyGroupTypes must match the aggregate types. UUIDs are also defined on an aggregate basis. The model maintains all aggregate-to-aggregate relationships.

   All keys added to the CRF table must have entries corresponding to applications requiring access to the data.

Appendix
Notices

This information was developed for products and services offered in the U.S.A. and Europe. IBM may not offer the products, services, or features discussed in this document in other 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
500 Columbus Avenue
Thornwood, NY 10594
U.S.A.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

IBM World Trade Asia Corporation
Licensing
2-31 Roppongi 3-chome, Minato-ku
Tokyo 106, Japan

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 the 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 United Kingdom Laboratories
Hursley Park
WINCHESTER, Hampshire
SO21 2JN
United Kingdom

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 information 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 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.

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

All IBM prices shown are IBM's suggested retail prices, are current and are subject to change without notice. Dealer prices may vary.

This information is for planning purposes only. The information herein is subject to change before the products described become available.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples 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.

and distribute these sample programs in any form without payment to IBM for the purposes of developing, using, marketing, or distributing application programs conforming to IBM's application programming interfaces.

Each copy or any portion of these sample programs or any derivative work, must include a copyright notice as follows:

If you are viewing this information softcopy, the photographs and color illustrations may not appear.

Trademarks

The following terms are trademarks or services of IBM Corporation in the United States or other countries or both:

IBM

MQSeries

DB2

OAG is a trademark of the Open Architecture Group in the United States or other countries or both.

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

Permission statement

Redistribution and use of this material for both commercial and noncommercial purposes are permitted subject to the below-stated conditions:

1. This Permission Statement shall be reproduced in its entirety in each copy of the material;
2. This material is provided AS IS without warranty of any kind, including but not limited to, any warranty of noninfringement or any warranty (express or implied) of merchantability or fitness for a particular purpose; and
3. The material may be modified provided
   a. Prior written notice of each modification is provided to the Interactive Financial eXchange Forum at the address listed below,

Interactive Financial Exchange Forum, Inc.
333 John Carlyle Street
Suite 600
Alexandria, VA 22314
U.S.A.

b. Any redistribution of modified materials shall be accompanied by a notice that modifications have been made and a clear description of the modifications, and

c. The party making the modifications assumes all responsibility for the consequences of the modifications.

Glossary

This glossary defines terms and abbreviations used in this book. If you do not find the term you are looking for, see the *Index* or the ***IBM Dictionary of Computing***, New York: McGraw-Hill, 1994.

A

Adapters
(1) A part that electrically or physically connects a device to a computer or to another device.

(2) A circuit board that adds function to a computer.

(3) Event Adapter: In a Tivoli environment, software that converts events into a format that the Tivoli Enterprise Console can use and forwards the events to the event server. Using the Tivoli Event Integration Facility, an organization can develop its own event adapters, tailored to its network environment and specific needs.

API: Application Programming Interface
(1) A software interface that enables applications to communicate with each other. An API is the set of programming language constructs or statements that can be coded in an application program to obtain the specific functions and services provided by an underlying operating system or service program.

(2) In VTAM, the language structure used in control blocks so that application programs can reference them and be identified to VTAM.

C

CRF: Cross Reference Function
This refers specifically to the storage system WMQI Enabler uses in order to keep track of creations of and attachments to UUID's.

D

DTD: Document Type Definition
The rules that specify the structure for a particular class of SGML or XML documents. The DTD defines the structure with elements, attributes, and notations, and it establishes constraints for how each element, attribute, and notation may be used within the particular class of documents. A DTD is analogous to a database schema in that the DTD completely describes the structure for a particular markup language.

M

MQMD: MQSeries Message Descriptor
The MQSeries Integrator (MQSI) header that contains basic control information that must travel with the message.

MQRFH
An architected message header that is used to provide metadata for the processing of a message. This header is supported by MQSeries Publish/Subscribe.

MQSeries
Pertaining to a family of IBM licensed programs that provide message queuing services.

MQSI: MQSeries Integrator
It provides graphical tools for constructing how critical data or business events are handled, by visually connecting a sequence of processing function to dynamically manipulate and route messages, combine them with data from corporate databases, warehouse in-flight message data for auditing or subsequent analysis, and distribute information efficiently to business applications.

P

Party
Any person or organization that the insurance company has, or had, or may have a business interest in.

Property
A data value of a type.

Q

Queue
An MQSeries object. Message queuing applications can put messages on, and get messages from, a queue. A queue is owned and maintained by a queue manager. Local queues can contain a list of messages waiting to be processed. Queues of other types cannot contain messages: they point to other queues, or can be used as models for dynamic queues.

S

System of Truth
This is a system that is accurate at all times. Any data that is added/change/verified comes from this system. The system of truth is defined for use by WMQI Enabler, which is the primary system or the system that would hold the most accurate data at any point in time for the systems attached to WMQI Enabler. It is regarded as the authority for any data being referenced and is the primary system for receiving any data updates.

U

UUID: Universally Unique Identifier
This is a key used by the WMQI Enabler to uniquely identify the entities which outside systems need to reference.

W

WMQI Enabler: WebSphere MQ Integrator Enabler
A complete scalable messaging and information integration add-on to the MQSeries family of products. Especially designed for the needs of the financial services industry, WebSphere MQ Integrator Enabler can integrate front-end systems with back-end systems using a hub/spoke architecture using XML as the common vocabulary across systems.

X

XML: eXtensible Markup Language
XML is a markup language for message definition, and is an open and public domain standard. XML is a subset of SGML designed for easy implementation in commercial and web environments.

Index