Retek® 10.1 Integration Bus

Technical Architecture Guide

Retek

The software described in this documentation is furnished under a license agreement and may be used only in accordance with the terms of the agreement.

No part of this documentation may be reproduced or transmitted in any form or by any means without the express written permission of Retek Inc., Retek on the Mall, 950 Nicollet Mall, Minneapolis, MN 55403.

Information in this documentation is subject to change without notice.

Retek provides product documentation in a read-only-format to ensure content integrity. Retek Customer Support cannot support documentation that has been changed without Retek authorization.

Corporate Headquarters:

Retek Inc.
Retek on the Mall
950 Nicollet Mall
Minneapolis, MN 55403

888.61.RETEK (toll free US)
+1 612 587 5000

European Headquarters:

Retek
110 Wigmore Street
London
W1U 3RW
United Kingdom

Switchboard:
+44 (0)20 7563 4600

Sales Enquiries:
+44 (0)20 7563 46 46
Fax: +44 (0)20 7563 46 10

Printed in the United States of America.

Customer Support

Customer Support hours:

Customer Support is available 7x24x365 via e-mail, phone and Web access.

Depending on the Support option chosen by a particular client (Standard, Plus, or Premium), the times that certain services are delivered may be restricted. Severity 1 (Critical) issues are addressed on a 7x24 basis and receive continuous attention until resolved, for all clients on active maintenance.

Contact Method	Contact Information
Internet (ROCS)	www.retek.com/support Retek's secure client Web site to update and view issues
E-mail	support@retek.com
Phone	US & Canada: 1-800-61-RETEK (1-800-617-3835) World: +1 612-587-5800 EMEA: 011 44 1223 703 444 Asia Pacific: 61 425 792 927
Mail	Retek Customer Support Retek on the Mall 950 Nicollet Mall Minneapolis, MN 55403

When contacting Customer Support, please provide:

- Product version and program/module name.
- Functional and technical description of the problem (include business impact).
- Detailed step by step instructions to recreate.
- Exact error message received.
- Screen shots of each step you take.

Contents

Chapter 1 – Introduction

Welcome to the Retek 10.1 Integration Bus Technical Architecture Guide. This guide describes the technical architecture of the Retek Integration Bus (RIB). The goal is to illustrate the capabilities and issues an enterprise may encounter when integrating applications with the RIB. The intended audience for this guide includes system designers and project managers. It assumes that you are familiar with Enterprise Application Integration terms and concepts. If not, see the "Additional resources" section for more information.

Chapter 2 introduces the RIB message model. Important conceptual topics are presented such as the business event relationship to the message, the message 'family,' and message structures. Because the sequence of events that occur on a table reflect business processes, this chapter discusses the association of message structure and sequencing to systems and their availability on the RIB. Error handling, performance, and the synchronization of participating systems are topics touched on here. Finally, Chapter 2 presents the message lifecycle, or how messages flow through the system. Described are simple flows of messages that do not require additional transformation, filtering, or routing logic (known as a 'TAFR') to occur on the RIB, and those flows that depend upon a further TAFR operation prior to another application's subscription of the message.

The components of both SeeBeyond's e*Gate Integrator–the RIB itself–and Retek applications on the RIB are described in Chapter 3. Here you learn about SeeBeyond components like the registry, schema, event type definitions, e*Ways, intelligent queues, collaborations, and more. In addition, this chapter presents components and processes shared by Retek applications on the RIB. Retek applications are characterized by the use of Oracle-based triggers and XML and message family manager packages for publishing messages through application adapters. Retek applications also share common message subscription processes for message and error handling. TAFR processing is presented too.

Learn about Retek message families in Chapter 4. The event type and message family concept is discussed. Here you can see a list of message families for each application: RMS, RCOM, and RDM. If you are considering the interface of additional applications on the RIB, read Chapter 5. The successful coupling of third-party applications to the RIB (and, as a result, to Retek applications) hinges on understanding the importance of the single event-message relationship. These concerns are addressed here, along with descriptions of SeeBeyond proprietary e*Gate adapters that a client can select for applications to be deployed on the RIB.

Chapters 6 and 7 introduce the integration of Retek Extract, Transform, and Load (RETL) and batch file transmission on the RIB. RETL (extraction-transformation-load) is a C++ framework that you can deploy within an application for high-volume processing, especially in a multi-processing environment. Both RETL and batch job integration involve the movement of files across the RIB. Currently, implementation of these processes involves further definition, and these chapters discuss the relevant issues.

Additional resources

Read the following Retek 10.1 and SeeBeyond documents for additional information.

Retek 10.1 integration documents

The following resources should be used for further understanding the Retek Integration Bus:

Retek 10.1 Integration Guide – Descriptions of Retek applications on the RIB and the functional areas in which they share data. The guide also contains all data descriptions, including the message catalog; XML document type definitions of messages; and mapping documents that specify a message's source application, table, column, and data type.

Retek 10.1 Integration Bus Primer – An introduction to basic Enterprise Application Integration (EAI) concepts and to the Retek Integration Bus (RIB).

Retek 10.1 Integration Bus Deployment Guide – Discussion of deployment considerations, design patterns, and strategies.

Retek 10.1 Installation Guide – Descriptions of:

- SeeBeyond e*Gate Integrator installation of its registry host and all participating host software, plus Graphical User Interface hosts for development and system monitoring.
- An explanation of how to import the RIB schema into the e*Gate Integrator product.
- Configuring database connection points and JMS queues, updating CLASSPATH configuration values, and deleting unused adapters.

Retek 10.1 Integration Bus Operations Guide–Provides a basic understanding of RIB components, how messages flow between them, and operational activities surrounding the components. Included are templates for using the RIB as an alternative to FTP batch jobs to transfer files from one system to another.

SeeBeyond Technology Corporation documents

See the resources listed in this section to learn more about the RIB as it is deployed through the SeeBeyond e*Gate Integrator EAI platform:

SeeBeyond Business Integration Suite Deployment Guide – Information to use in analyzing, planning, and managing an EAI deployment.

SeeBeyond Business Integration Suite Primer – An introduction to all SeeBeyond e*Gate products, including e*Ways for popular applications like:

- PeopleSoft
- SAP
- Oracle Financials

Chapter 2 – The RIB messaging model

This chapter presents the RIB's messaging model. It describes how RIB messages are structured and the rationale behind this structure. It also describes the types of messages used.

Not presented in this chapter are the specifics of each message. The Retek 10.1 Integration Guide details information about message contents and transformations.

Message characterization

Enterprise Application Integration systems produce messages characterized by three dimensions: the contents of the message, when the message is produced, and the structure of the message.

Note: The term "message characterization" is used as opposed to "message type" to avoid confusion with other EAI terms.

Structure: The message may have a simple structure and correspond to a small business sub-entity or it may contain a hierarchical structure containing all sub-entities that comprise it. ("Flat" versus "hierarchical".)

Message contents: The message contains all information about a business entity or it captures only something that has changed about that entity ("snapshot" versus "delta").

When the message is produced: The message may be produced as part of the business transaction affecting the entity or it may be produced within a separate transaction that occurs a short period of time later. ("Synchronous" versus "asynchronous" production.)

Using these criteria, one is able to characterize a specific message as a "flat synchronous snapshot" or a "hierarchical asynchronous delta" or a "hierarchical synchronous snapshot" or some other combination. Additional information accompanies the business entity information. This includes XML tags used to rout the message, information about the originating system or environment, or information about the business event the message captures.

The RIB publishes three different message characterizations:

- Hierarchical Synchronous Snapshots – These messages contain newly created composite business entities, such as purchase orders.
- Flat Synchronous Snapshots – These messages contain a change made to a business entity absolute value, such as the price of an item, on a "master" system. They may also contain newly created simple business entities, such as a location.
- Flat Synchronous Deltas – these messages encapsulate a business event captured on a non-master system that affects information on a remote "master" system. An example of this would be for a clerk to reserve inventory for a local store system from a remote warehouse system. The remote warehouse system is the master of its inventory data.

RIB message families and message types

Besides the characterizations of a message, each RIB message belongs to a specific *message family*. Each message family contains information specific to a related set of operations on a business entity or related business entities. The publisher is responsible for publishing messages in response to actions performed on these entities in the same sequence as they occur.

Descriptions of each message family are found later in this document. Although a generalized format exists, each message family varies in the specifics of the information it contains – the business entities and events the message captures. Furthermore, each message family contains a set of sub-formats specific to the business event triggering message publication. The term *message type* embodies this specific sub-format. For example: a Purchase Order message family can contain message types such as “Create PO Header”, ”Create PO Detail”, “Update PO Header”, or “Delete PO Detail”.

Messages are published and subscribed to on a message family basis. A single application is responsible for publishing all messages within a message family. However, multiple instances of an application may publish messages within the same message family. In other words, only the RMS application publishes messages in the “Available To Promise” (ATP) message family and only the RDM application publishes messages in the “Advanced Ship Notice Outbound” (ASN Outbound) message family. However, multiple distribution center installations of RDM may each publish their own ASN Outbound messages.

Model drivers and concerns

An architect chooses the type, structure, and other characteristics of the messages within an EAI system based upon many factors. One major factor is that the message contents encapsulate the business event. Different types and characterizations are available within a single EAI system. The RIB is no exception. The RIB contains many messages corresponding to the creation of an entity characterized as "Hierarchical snapshots" and "synchronously" produced. On the other hand, there are also "flat synchronous delta" RIB messages associated with update operations. The factors determining which characterization to use include:

- **Publisher/subscriber/bus availability:** One major goal in the design of the RIB is to insure that no tight coupling exists between Retek's applications and the RIB's availability. That is, if the RIB is unavailable, the publishing and subscribing applications can still function. This means that there may be a delay before the transmission of a message occurs over the RIB network. It also means that database updates needed for message publishing must occur outside of the same transaction containing the business event.
- **Retek application locking on sub-business entities:** Many of Retek's applications allow for simultaneous updates to sub-business entities. An example of such an entity is a line item found within a Purchase Order. The Retek Merchandising System allows multiple concurrent changes to multiple items, header, or summary information for a single PO. Many times the PO is used for replenishment purposes and multiple people are constantly updating the PO. Situations such as these tend to produce "flat" messages containing only the changes to the line items. Producing a "hierarchical" message would risk locking the PO for an unacceptable amount of time.
- **Concurrency of message contents production and business event:** A desire for a loose coupling between the EAI system and the business application suite drives some EAI architectures. In many cases, message information is staged before publication. A delay exists between when the business event occurs and when the message corresponding to this event is created and published. This delay presents a window of opportunity for multiple similar business events to occur on the same entity before publication of any of the messages. For example, multiple users may make changes to the same Purchase Order header within a short time period.

  There are two strategies for staging business event information: record only enough information to denote that the event occurred (for example: an update occurred on PO line item xxx) or record all information about that event (for example: an update occurred on PO line item xxx and the new quantity is yyy, the new location is zzz,). If all information about the event is NOT staged, the message published may not correspond to the triggering business event. In an even worse scenario, it's possible to delete the business entity involved before creating the message.
- **Transactional considerations:** Some business events require multiple database transactions to complete. One example of this is the creation of a new vendor.

- **Sequencing and error handling:** Many business processes are stateful. That is, only certain actions can occur at certain times. A subscriber must process messages concerning a specific business entity in the same order they were published. This has implications regarding error handling: once an error occurs on one message, subsequent messages referring to the same business object should be held and not processed until the error has been resolved. However, other messages concerning other business entities should continue to be processed.
- **Deployment and software lifecycle:** The applications producing and subscribing to messages need separate deployment between themselves and the RIB. In effect, each Retek application can be "plugged" into the RIB based on the needs of the retailer. If the retailer decides to not use the RIB, then no noticeable performance degradation occurs. In other words, the RIB is not required for any Retek application.
- **Performance:** Updates to some business sub-entities happen frequently on a single business entity. Take the example of a retailer creating a single replenishment PO per supplier. Users may update the same PO many times during the day. When one analyzes the volume of updates and the cost of creating a full PO message, it may be a significant performance bottleneck to publish the full PO snapshot for each update.
- **Data synchronization risks:** Many messages seek to replicate data across multiple systems. Sometimes, the data on two systems may differ due to a variety of possible situations. When one uses a "delta" type of message, there is a risk that the subscriber cannot process these messages due to the data differences.

Message life cycle

The Retek Information Bus (RIB) uses the "Pub/Sub" message model for all of the messages produced and consumed within the EAI system. The publishing application is responsible for creating the initial message contents. The RIB publishing adapter will publish it to the RIB and make it available to any subscribers. The RIB knows what subscribers are to receive the message due to the RIB's configuration -- this configuration associates a set of subscribers to each publisher / message family combination.

Database tables associated with the publishing application typically stage message information. One or more RIB Publishing Adapter collaborations poll the staging table. A collaboration is a single thread of control within the adapter that publishes the appropriate XML message(s) to the EAI network. Each publishing collaboration publishes messages that are specific to a predetermined subset of business entities.

Note: The actual subset of business entities is determined by configuration entries. For the Purchase Order message family, for example, if two collaborations are configured, then one may publish the even numbered POs and the other the odd numbered POs. Note: not all RIB adapters have this capability.

The message resides in a network queue immediately after publication. This queue provides stable storage for the message in case of a system crash occurring before all message destinations receive and process it. One requirement for the queue is that it must be delivered and processed successfully exactly once to each subscriber. All work performed by the subscriber and the RIB must be atomically committed or rolled back, even if the EAI queue is on a remote host using a remote database. The standard way to perform this is by using an XA interface and two-phase commit protocol.

After initial publication, a message may undergo a series of transformation, filtering, or routing operations. A RIB component that implements these operations is known as a Transformation and Address Filter/Router (TAFR) component. A *transformation operation* changes the message data or contents. A *filter operation* examines the message contents and makes a determination as to whether the message is appropriate to the subscriber. For example: those subscribers that do not process all message types found in a message family require filter operations to weed out the unsupported types. A *router operation* examines the message contents and forwards the message to a subset of its subscribers. A filter operation can be considered a special case of a routing operation. Although logically separate operations, for performance reasons TAFR components usually combine as many as is appropriate.

TAFR operations are specific to the set of subscribers to a specific message family. Multiple TAFRs may process a single message for a specific subscriber and different specific TAFRs may be present for different subscribers. Different sets of TAFRs are necessary for different message families.

If all subscribers to a message can process all messages within a message family without any TAFR operations, then no TAFR components are needed, as seen in Figure 2.1. However, multiple TAFRs may be needed depending on the types of subscribers. This is seen in Figure 2.2, where one TAFR routes the information among different remote sites and then another TAFR transforms the data further for an additional subscriber.

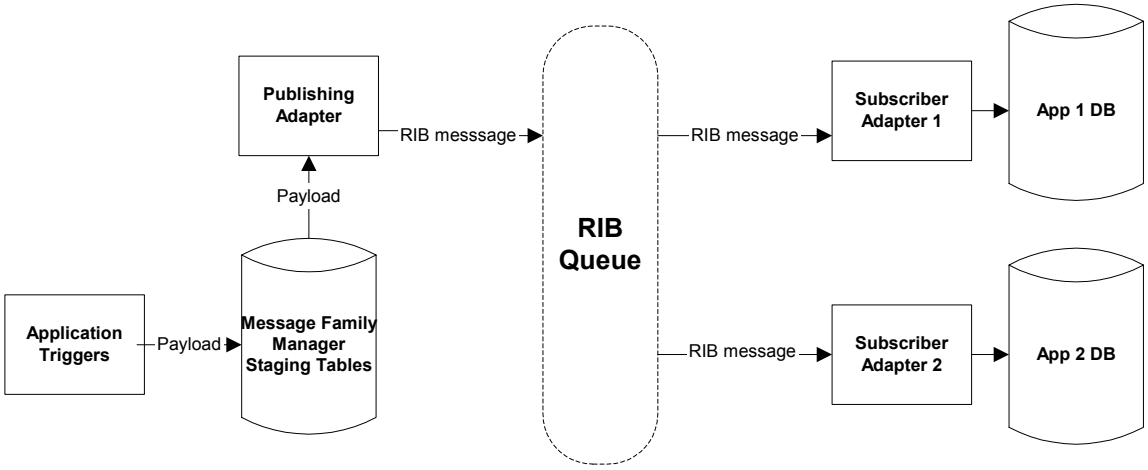


Figure 2.1 Simple Message Flow

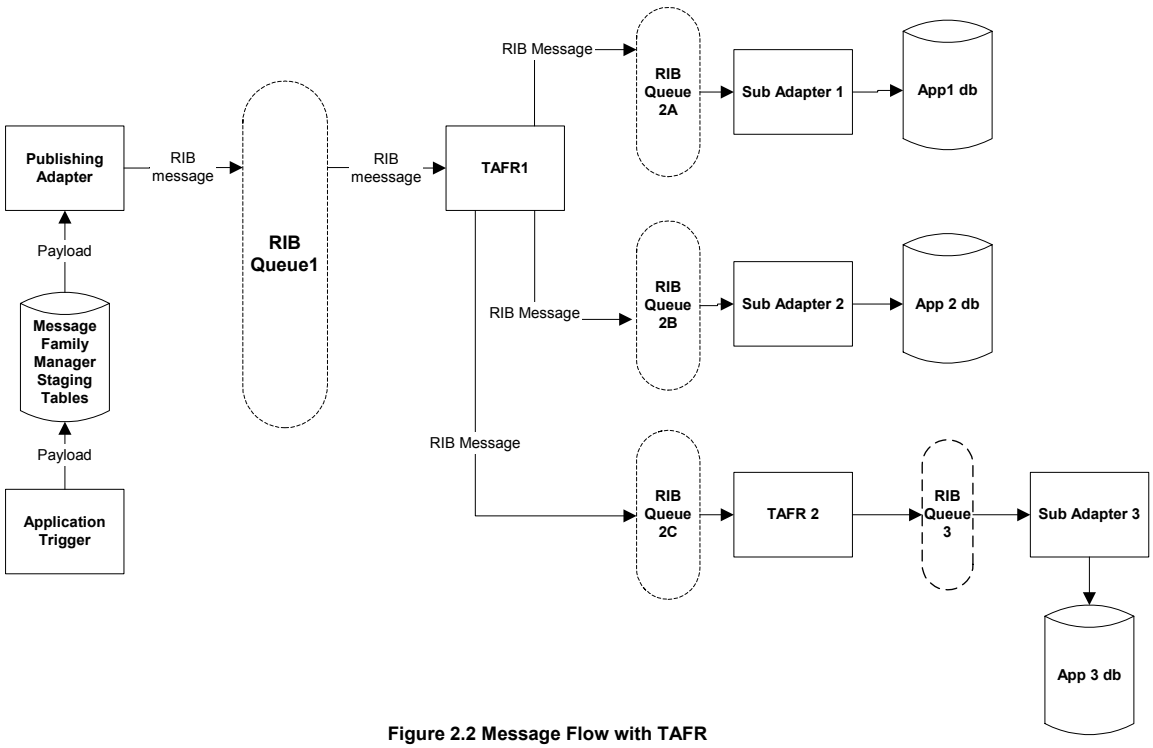


Figure 2.2 Message Flow with TAFR

Another type of RIB component that may process a message is a *bridge* component. These SeeBeyond e*Ways, BOBs, queues, or connection points allow messages to traverse different administrative domains. The type of bridge component used is site specific. A deployment of bridge components is dependent on how the network bandwidth and topology, the administrative specifics of the publisher and subscriber applications, and the availability of specific RIB resources. Bridges are very useful when remote sites that belong to different organizations and operations staff need to exchange messages and a central controlling authority is non-existent. Figure 2.3 is a modification of Figure 2.2, where one of the remote systems uses a bridge.

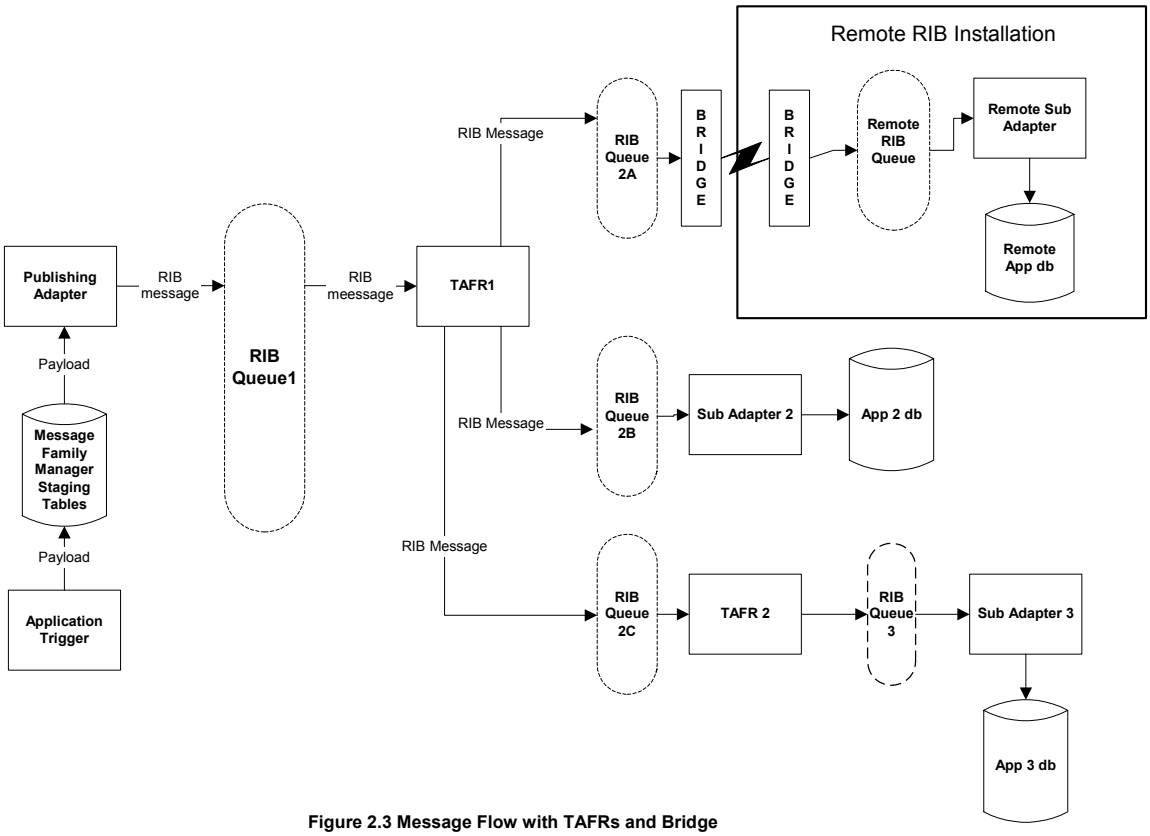


Figure 2.3 Message Flow with TAFRs and Bridge

Within RIB components, message processing continues until a subscribing adapter successfully processes the message. These components will perform application specific database updates for the specific message encountered.

When a message is processed, the adapter checks to see if the message hospital contains any messages associated with the same entity as the current message. If so, then the adapter places the current message in the hospital as well. This is to insure messages are always processed in the proper sequence. If proper sequencing is *not* maintained, then the subscribing application will contain invalid data.

If an error occurs during message processing, the subscribing adapter notes this internally and rolls back all database work associated with the message. When the message is re-processed (since it has yet to be processed successfully), the adapter will now recognize this message is problematic (sick) and checks it into the hospital for “surgery”.

After a message is checked into the Error Hospital, a second collaboration extracts the message from the hospital and re-publishes it to the integration bus. The message remains in the hospital during all re-tries until the subscribing adapter successfully processes it or the maximum allowed retries is reached.

RIB message structure

RIB Messages are XML formatted and have a two-tiered structure consisting of a set of “envelope” tags and a single “payload”. The envelope tags contain routing, message type, and other non-business entity information. The payload is specific to the message type and contains the business entity information.

As of the RIB 10.1 release, the message envelope contains the following tags:

`<family>` message family message belongs to

`<type>` message type message belongs to

`<id>` Optional ID string that identifies the message. Composite primary keys will require multiple IDs. For example, a line item within a Purchase Order would contain the PO number and line item number as part of the ID. For example:

```
<id>PONumber=12345</id>
<id>ItemID=321</id>
```

Some ID’s are simple and the value of the ID is specific to the message family. In this case, a single ID tag may be present and consist of merely a single identifier, such as

```
<id>FT_ITEM_12</id>
```

`<routingInfo>` Optional tag that contains elements used to route or filter messages for specific subscribers. Multiple <routingInfo> tags may be present. Within the <routingInfo> element, the following sub-elements must exist:

- `<name>` name of routing field. A message may have multiple routing fields.
- `<value>` value of the routing field.

`<publishTime>` Date/timestamp the message was published.

`<hospitalID>` ID of the Message within the Error Hospital. Set only after the message is checked into the Error Hospital.

`<failure>` Optional tag that contains elements used to identify a specific processing error. Multiple <failure> tags may exist. Every time the message is checked into the Error Hospital, a <failure> tag is created. This tag contains the following sub-elements:

- `<time>` Date/timestamp of failure.
- `<location>` Location or name of the Error Hospital.
- `<description>` Textual description of the error.

`<messageData>` The message type specific “payload” containing data describing the message triggering event. The payload is XML, but the XML varies within each message type. The DTDs describing this data are stored in a table within the rib_messages database table.

`<ribmessageID>` This field uniquely identifies the message based on the publishing adapter. It may be used to track or correlate problems assocaited with a specific message.

Chapter 3 – Messaging system component overview

This chapter details the major components of the RIB that create, process, or consume messages.

SeeBeyond components

Most of the RIB components execute within the context of SeeBeyond's e*Gate Integrator environment. This section presents a brief overview of these components.

Registry

The e*Gate Registry is a SeeBeyond proprietary database containing all entities used within a running e*Gate system. There is at least one registry available to SeeBeyond components at all times. A system designer designates one registry as the "master". Other, "secondary" registries replicate the master for increased performance and system availability.

Schemas

A schema is a logical grouping of EAI components. Each registry contains at one or more schemas. Typically, schemas are designed for the end-to-end processing of a set of related messages. The design of a Schema within a deployed RIB system is dependent on many site-specific factors. Specific design or configuration options are discussed in the RIB Deployment Guide.

Control brokers and participating hosts

The control broker is responsible for maintaining the operational control and status of its attached components. Another goal of a control broker is to minimize the number of network connections to the registry and to provide a central point of control for a set of components. Each control broker connects to a registry but can also fail over to other registries if needed. The control broker and all of the attached components must belong to a single e*Gate schema.

There is one control broker per "participating host" per schema. A participating host is a logical construct used. The control broker's TCP/IP address and the participating host's name are associated with each other within the registry.

Control Brokers and participating hosts are transparent or not involved in the processing of RIB messages.

Events and event type definitions

SeeBeyond "events" include both messages passing to and from JMS, and stored procedure calls to external application APIs. An event's type determines its logical name, and its physical structure is determined by an event type definition (ETD). Different event types may share the same ETD to allow message with identical structure to flow to different recipients. The RIB uses a single ETD for all messages while they are inside the RIB.

Collaborations

Collaborations define message processing logic on a per message family/message source/ component combination. This logic is "triggered" or executed when the adapter pulls a message with the correct event type from the specified source. The RIB uses Java to define the message processing logic. All collaborations require one or more triggering conditions in order to execute. This condition may be any of the following:

- A file appearing in some directory
- A certain time period has elapsed
- A message appearing on a queue
- Some application – specific condition

A collaboration works on a collection of input and output events, which may be messages going to or from queues, or passing to or from an application's RIB APIs.

In general, the logic within a collaboration may perform any number of operations. It may update a database, simply collect statistical data, write information to a file, or some other operation. It may produce zero or hundreds of output events, depending on the application.

e*Ways and BOBs

There are two basic types of e*Gate components used to create, process, and/or consume messages on the RIB: e*Ways and Business Object Brokers (BOBs). These are specific implementations of the generalized concept known as an Integration Bus "Adapter". BOBs and e*Ways contain one or more "Collaborations" that are triggered from some event. A collaboration works on a collection of input and output events, which may be messages going to or from queues, or passing to or from an application's RIB APIs.

Note: See the Retek Integration Bus Primer if you are unfamiliar with the concept of an Integration Bus Adapter.

e*Ways and BOBs are multi-threaded and can process multiple messages simultaneously, but are single-threaded for a particular event type.

Traditionally, the difference between the two component types is that e*Ways may contain an "application specific" source or sink for messages, while BOBs connect internal bus components. The RIB, however, only uses a specific type of e*Way, the Java "Multi-mode" e*Way, which can function as both an external source or sink and an internal connector. The Multi-mode e*Way is a grouping of logical collaborations into a single physical process or program.

Intelligent Queues and JMS Intelligent Queues

Intelligent Queues (IQ) hold published messages and maintain a record of what subscribers have received the messages. Many types of Intelligent Queues either wrapper the message storage mechanism or bridge to another queuing system. The SeeBeyond e*Gate system installed with the RIB includes the standard file-based IQ, a Java Messaging Service (JMS) IQ, and an Oracle IQ. Also available is a memory-resident queue used to store messages sent between collaborations that are entirely resident within a single e*Way.

JMS Intelligent Queues are queues that may be accessed using the Java Message Service API. All supplied RIB Queues use connection points defined for this API.

IQ Managers and JMS IQ Managers

The original purpose of an Intelligent Queue Manager was to control a set of Intelligent Queues of the same type. There are multiple types of Queue Managers, each controlling a different type of IQ. Each type of IQ differs on how messages are queued and saved to stable storage while in the queue.

The JMS Intelligent Queue Manager serves two roles. The first is the same as any other IQ manager: to control a set of Intelligent Queues for any SeeBeyond e*Way. The second (which the RIB uses) is to act as a Java Message Service (JMS) provider, accessible through JMS Connection Points. The RIB uses the IQ Manager this way because it requires the use of the XA two-phase commit protocol to guarantee "exactly once" successful message processing. This protocol is available with a JMS implementation. However, a JMS Intelligent Queue is not used because the existing IQ Manager service interface does not support the XA protocol. Instead, RIB e*Ways use SeeBeyond JMS Connection Points. Connection Points connect to a JMS IQ Manager such that the XA protocol is supported. For more information regarding JMS connection points and Intelligent Queues, see the *SeeBeyond JMS Intelligent Queue User's Guide.*

e*Way Connection Points

An "e*Way Connection" or "Connection Point" defines a session between the e*Way and an external system. The following types of connections are available:

- Java Message Service – a connection to a JMS Server or JMS Service.
- A relational database, such as Oracle
- A TCP/IP connection to a remote application using the HTTP or HTTPS protocol.
- E-mail (uses SMTP for outbound and POP3 for inbound messages)

If a Database connection point used within a collaboration defines the login, password, and server address for database operations. It also may define the frequency "triggering events" are fired off, allowing the collaboration to define a polling operation.

A connection point made to a JMS implementation can be used to publish or subscribe to external applications. JMS connection points can also be used to bridge between e*Gate schemas.

RIB components

The SeeBeyond components listed above build and process RIB messages. This section lists the subsystems deployed within these components and within other Retek application software. Each RIB component has a dedicated task and is generally specific to one message family.

RIB_XML database package

Because so much of the RIB uses Oracle XML, Retek developed utility and helper procedures. These procedures are stored within the `RIB_XML` PL/SQL package. This package simplifies the process of creating new XML strings or parsing existing XML strings.

Message validation: The RIB_XML package can perform message payload validation against a Document Type Definition (DTD). This DTD is stored as a CLOB within the database. If the publishing or subscribing application requests validation, the RIB_XML package API contains parameters to extract the DTD from rib_doctypes, parse the DTD and then validate the message payload using the DTD.

The table `rib_doctypes` stores the DTD as a CLOB and associates the CLOB with a message name. This table must be accessible within the user ID used to create or consume RIB messages. Loading the rib_doctypes table may be performed using the *DocTypeInserter* java application.

RIB_SXW database package

Another Oracle package has been developed for creating XML payloads, the RIB_SXW package. This package provides no validation facilities, but better performance than RIB_XML. It also does not contain any parsing functions.

This package also contains restrictions in how a message may be created, such as fully populating an XML element with fields and sub-elements before moving to another node on the XML tree.

Application message publishing triggers

Most applications use triggers to initiate the message publishing process. These triggers are RIB specific and should be enabled only when an enterprise is using the RIB for integrating its applications. These triggers are fired when a specific database table is modified. The trigger assumes that the application is responsible for the modified data. The trigger retrieves all of pertinent information to create a specific type of message and inserts it into a staging table using an application specific Message Family Manager (MFM) API.

The message information is usually stored as an XML string and is known as the RIB message "payload". The payload is contained in an Oracle Character Large Object Binary (CLOB). The database table that holds the payload data must also maintain the following:

- The order that messages are created
- The CLOB containing the "payload" XML
- Any routing or filtering key values
- The message type associated with the business event that created the message. The message type is specific to the message family and a single business event may produce multiple messages of differing types within different families.

By storing all of the data within the same transaction as the business event, all RIB messages are considered as being "published" synchronously with the business event – even though the message has not been processed by any EAI system deployed component.

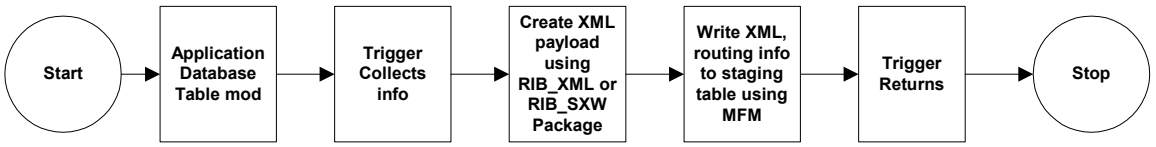


Figure 3.1 Trigger Processing -- XML CLOB

Figure 3.1 displays the application trigger processing. The following steps are followed:

1 An insert/update/delete operation on a table causes a RIB application trigger to be executed. The trigger was installed and enabled as part of the RIB installation.

2 The trigger collects any information it needs to continue. This may involve additional database operations.

3 The trigger leverages either the RIB_XML or RIB_SXW package to buld the XML payload for this message type. An Oracle CLOB is created to store the XML payload.

4 The trigger calls the Message Family Manager package to store the message into a staging table. The specific API that is called is the ADDTOQ() procedure.

5 The trigger returns.

Non-trigger publishing

Some applications may not use triggers to start the publishing process. Some alternatives used are:

- Using an insert into the MFM staging table directly from Oracle Forms. In this case, the logic to create the CLOB and insert it into the MFM staging table is found in a stored procedure referenced directly by the Oracle Forms based application.
- Using "upload" tables to stage the information until ready to publish. In this scenario, the message is not bound to the XML format until the Message Family Manager GETNXT() stored procedures invoked. GETNXT() is described in the next section.
- Using a file to create the RIB Messages. This would typically be used for interfaces from external systems.

In first two cases above, the information contained in the message published to the bus is stored within the same transaction as the business event. The actual technique used to kick off a message's publication is described in more detail in the Retek 10.1 Integration Guide.

Message Family Manager API

Each application uses a Message Family Manager (MFM) specific API for publishing all messages within a specific message family. This API is the interface to a stored procedure package and wrappers the staging table and additional business logic surrounding the message publication. A single application is responsible for publishing all messages within a single MFM.

Because the same application can publish multiple message families, it could use multiple MFM specific packages, one per MFM.

There are two procedures typically included in an MFM package:

ADDTOQ()

Stores message state, routing / filtering keys, message type, XML Payload, and other information needed to create a RIB Message. This procedure has the following format for its parameter footprint:

```
PROCEDURE ADDTOQ( O_status_code      OUT   VARCHAR2,
                  O_error_text       OUT   VARCHAR2,
                  I_message_type     IN       VARCHAR2,
                  I_message          IN       CLOB,
                  I_msg_1            IN       tbl.msg_spec_1%TYPE,
                  …

);
```

where

O_status_code	Denotes the status of the call. The value of this is found in the RIB_CODES package. Possible values include: MFM_FATAL_ERROR – cannot insert a message due to an error. MFM_SUCCESS – successful message insertion.
O_error_text	This is text associated with an error or warning occurring in the call to ADDTOQ.
I_message_type	Type of the message payload. A specific type is associated with one or more business events. This type is a further subdivision of the message family.
I_message	The message payload formatted as an XML string.
I_msg_1	A message family specific facility type, key, or other information that is supposed to be present in the message envelope. This is an optional parameter and may not be present. The type of this parameter is specific to the message family.
…	Additional optional parameters. These are dependent on the message family in use.

GETNXT()

Retrieves the record from the staging table for publication. This procedure uses the following parameter signature:

```
 PROCEDURE GETNXT( O_status_code     OUT   VARCHAR2,
                   O_error_text      OUT   VARCHAR2,
                   O_message_type    OUT   VARCHAR2,
                   O_message         OUT   CLOB,
                   O_msg_1           OUT   tbl.msg_spec_1%TYPE,
                   …
);
```

where

O_status_code	Denotes the status of the call. The value of this is found in the RIB_CODES package. There are for possible values: MFM_FATAL_ERROR – cannot retrieve a message due to an error. Publisher should exit. MFM_WARNING – the next message cannot be published because of a sequencing problem. MFM_SUCCESS – successful message retrieval. MFM_NO_MSG – no messages are waiting to be put onto the integration bus.
O_error_text	Text associated with an error or warning.
O_message_type	Type of the message payload. A specific type is associated with one or more business events.

`O_message`	The message payload formatted as an XML string.
`O_msg_1`	A message family specific facility type, key, or other information that is supposed to be present in the message envelope. The Type of this parameter is specific to the message family.
...	Additional optional message family specific parameters.

Publishing application adapter

Most messages are published using two separate database transactions, as seen in Figure 3-2. The first transaction consists of the application specific insert/update/delete operations that perform some business functionality. These operations occur independently of the RIB. However, when the RIB is active, additional triggers are enabled on these tables that create one or more XML "payloads" that are inserted into one or more staging tables. These "payloads" encapsulate the business event the application is performing.

Access to the staging table is wrappered by a Message Family Manager (MFM) PL/SQL package. Many MFMs exist and more detail on them will be found later in this guide. Each publishing MFM contains a stored procedure used to insert into the staging table is called "AddToQ()". This function always has an input CLOB parameter that contains the XML Payload. It may also contain a variable number of additional parameters.

The second transaction is controlled by the publishing adapter. A RIB Publishing Adapter polls the staging table by calling another routine in the MFM called "GetNxt()". This type of operation is known as a "Pull", since the adapter pulls the data from the database. The MFM "GetNxt()" procedure may contain simple or complex logic, depending on the messages it is responsible for. For example, a simple "Create Vendor" message may involve merely selecting and then deleting a single record from the vendor staging table. On the other hand, a "Create Purchase Order" message has more complex logic, since one does not publish purchase orders until they have been approved. Many changes may be made to a PO before it is approved and each change resulted in a call to the MFM's AddToQ() function to add another record into the staging table.

When the call to the MFM GetNxt() returns the data to the publishing adapter, a RIB Message is created from the payload (and other) GetNxt parameters. This message is then published to a Java Message Service (JMS) Topic (also called a "RIB Queue").

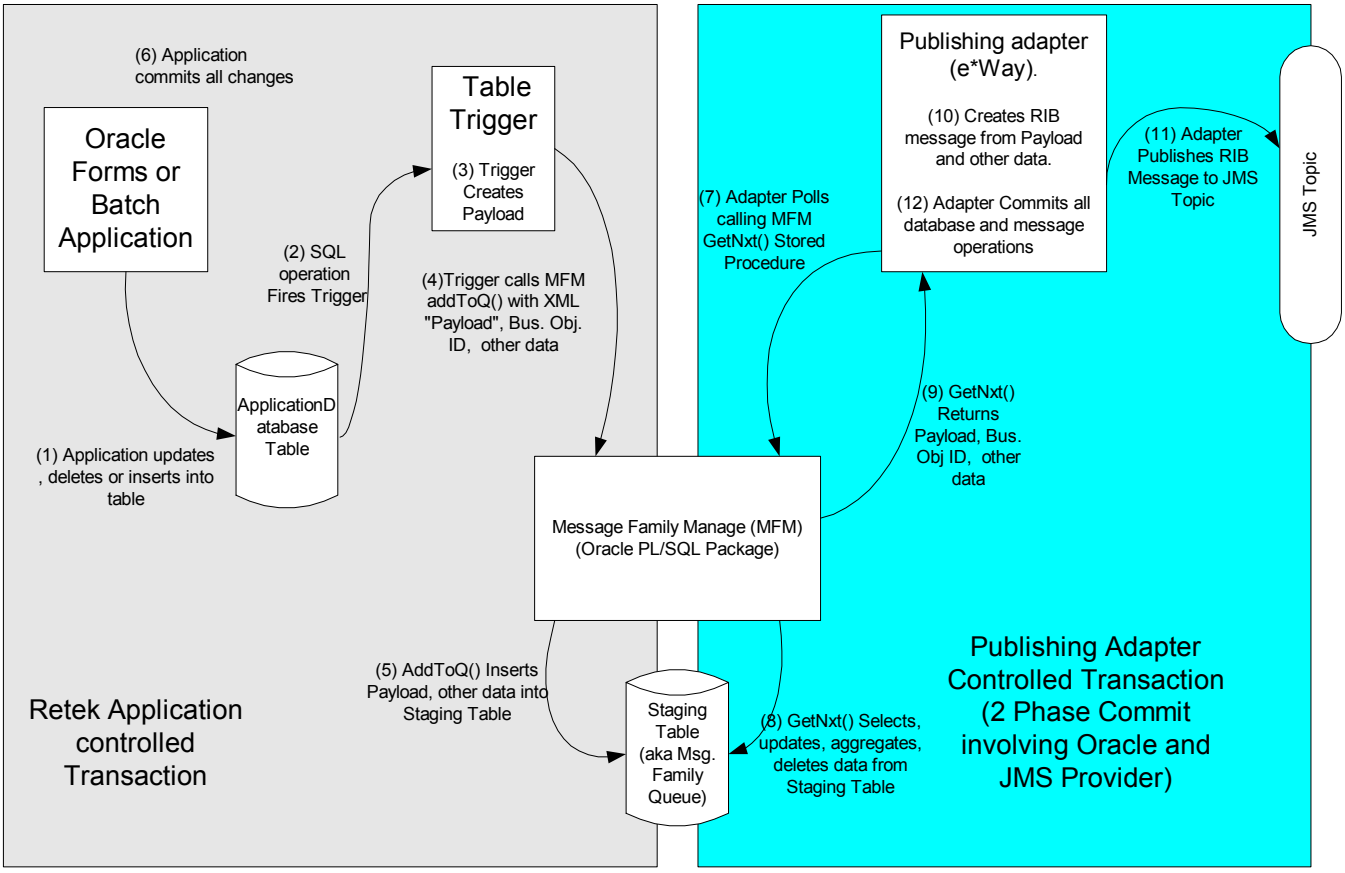
Note: In the Java Message Service nomenclature, one puts a message onto a JMS "Topic" for Pub/Sub operations. One puts a message onto a JMS "Queue" when only a single subscriber will ever receive the message.

An XA compliant two-phase commit operation is then performed to insure that all operations on the database and the RIB Queue are performed atomically. I.e. the data is either deleted from the database **and** published to the RIB Queue, or neither deletion **nor** publication occurs.

Note: XA is a Distributed Transaction Processing specification originally developed in 1991. It is now available from "The Open Group". Copies of this standard (*C193 Distributed TP: The XA Specification* ISBN 1-872630-24-3) are available from "The Open Group's" website, http://www.opengroup.org.

As long as the GetNxt() procedure returns a message, the publishing adapter will immediately publish the message and make another call to GetNxt(). If GetNxt() returns a "No message available" status, the publishing adapter will sleep a configured amount of time before it tries to call GetNxt() again.

The message resides in a network queue immediately after publication. This queue provides stable storage for the message in case of a system crash occurring before all message destinations receive and process it.



Message Publication Process

Figure 3-2

TAFR Adapter

A Transformation Address Filter/Router (TAFR) adapter is another e*Way adapter that is used to process data. It contains one or more collaborations that perform TAFR operations on all messages from a single message family. The specific activities it performs is dependent on the needs of its subscribers.

Figure 3.3 illustrates the activities associated with a TAFR adapter. These include:

1. A message is delivered to the TAFR adapter collaboration after it has been placed onto a JMS queue. This triggers the collaboration logic.
2. The TAFR performs its needed filtering and transformation processing on the message.
3. If the message is to be routed to one or more destinations, the message contents are copied into a new SeeBeyond Event Type. This event type is specific to the destination. Hence, if an Advance Ship Notice Inbound message needs to go three different warehouses, then the full contents of the message is published to the integration bus as three different events using three different event types. This allows for each of these messages to be published to different queues.

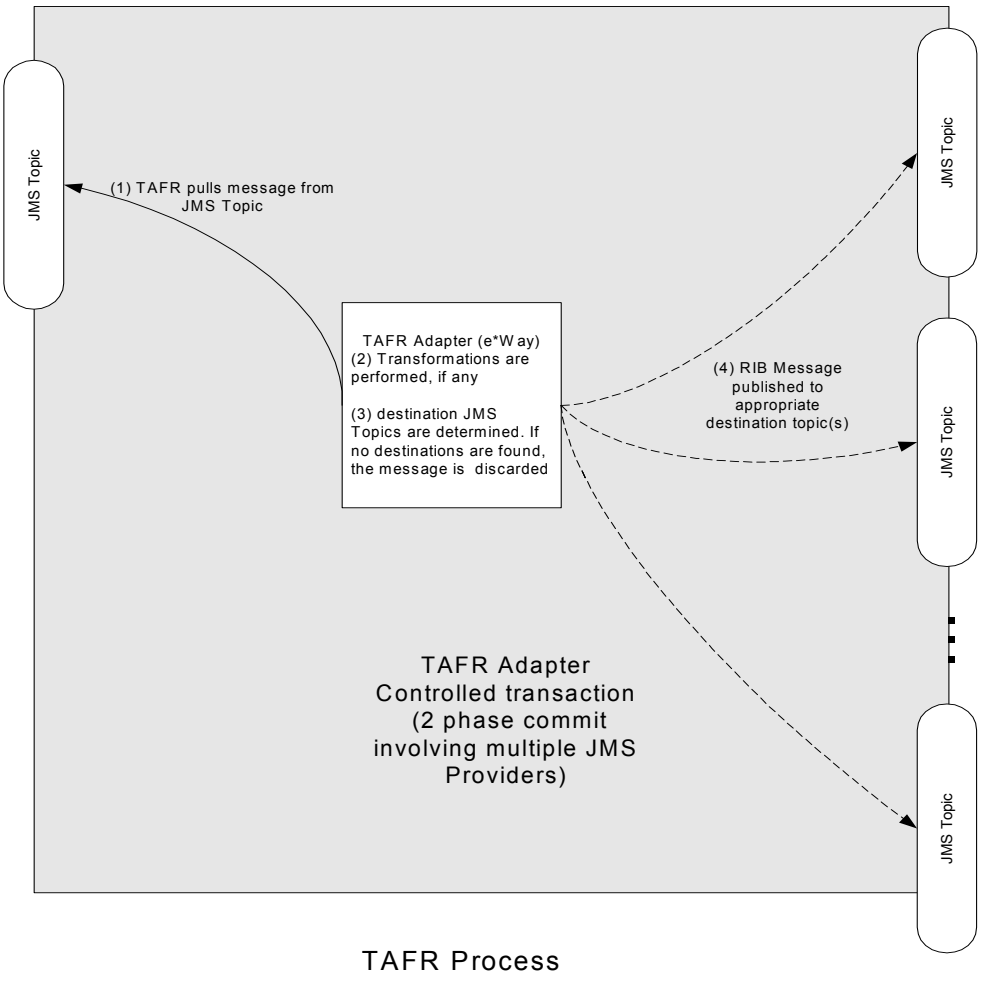

JMS Topic
(1) TAFR pulls message from JMS Topic
TAFR Adapter (e*Way)
(2) Transformations are performed, if any
(3) destination JMS Topics are determined. If no destinations are found, the message is discarded
(4) RIB Message published to appropriate destination topic(s)
JMS Topic
JMS Topic
JMS Topic
TAFR Adapter Controlled transaction (2 phase commit involving multiple JMS Providers)
TAFR Process

Figure 3-3

Subscribing application adapter

An application subscribes to a message through the use of a RIB subscribing adapter. This adapter will pull a message from the appropriate JMS topic and call a subscribing MFM's "Consume()" stored procedure. For subscribers, there may be many MFM PL/SQL packages for a single Message Family, each one with offering its own "Consume()" procedure. The purpose of this procedure is to directly update the application controlled tables with the information found in a specific RIB message type.

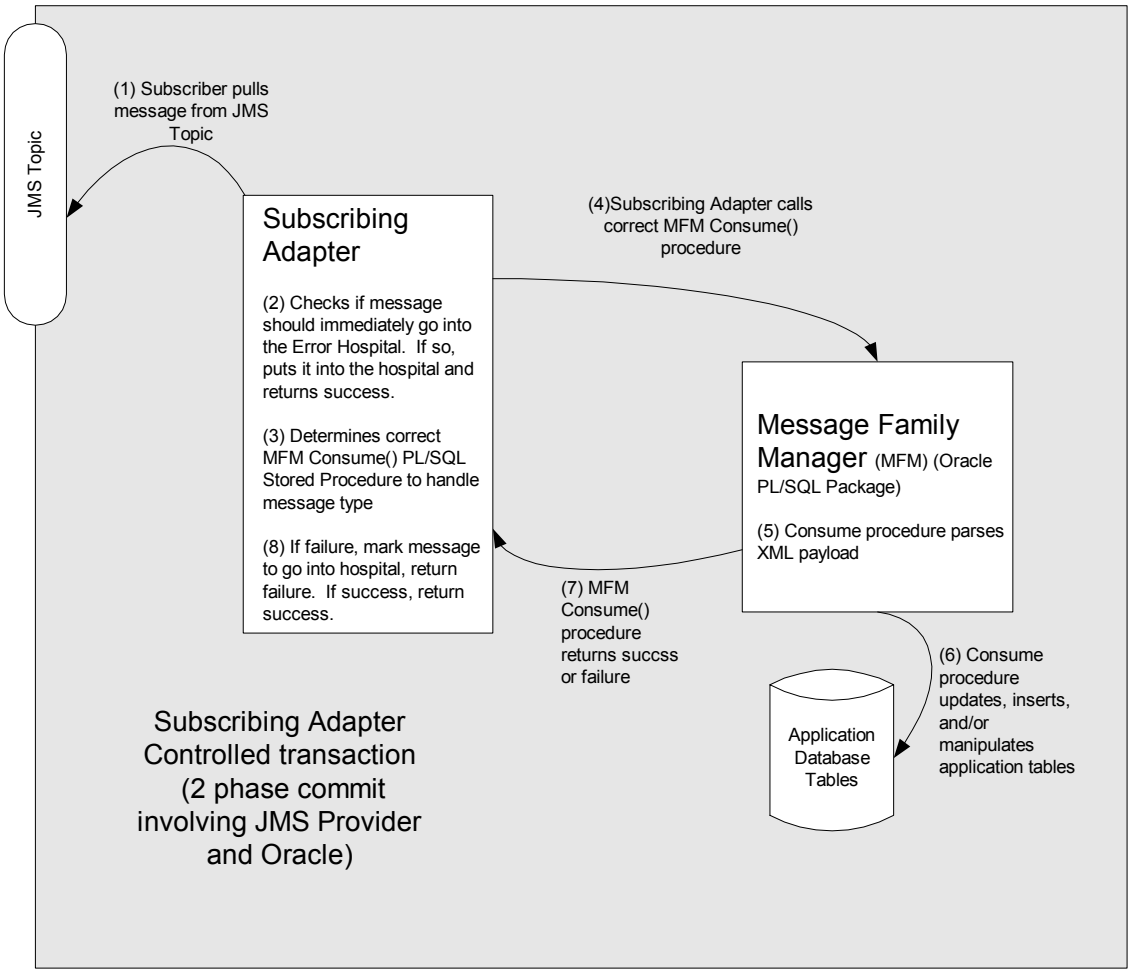
Subscribing adapters are also responsible for insuring that messages are processed in the correct sequence for a given business entity. For a specific Purchase Order, its "Create Purchase Order" message must always be processed before an update or delete message. Furthermore, all updates must be processed in the correct order to insure that two systems are correctly synchronized.

But no such guarantee exists when comparing messages concerning different business entities. In general, messages are processed in a First-In, First-Out order. If an error occurs processing a message for a PO, then other messages for other PO's should still be processed.

If an error occurs during message processing, the subscribing adapter notes this internally (NOT in the database) and rolls back all database work associated with the message. When the message is re-processed (since it has yet to be processed successfully), the adapter will now recognize this message is problematic (sick) and checks it into an Error Hospital database for "surgery".

A subscribing adapter always checks the hospital database to see if there are any messages in the hospital that act on the same business entity (such as a PO) that the current message does. If so, then the adapter places the current message in the hospital as well. This is to insure that all messages for a given business entity are processed in the correct order. Without manual intervention, the RIB will always process the "Sick" messages for a business object before any subsequent messages that act on the same business object.

After a message is checked into the Error Hospital, a second thread of control within the adapter extracts the message from the hospital and re-publishes it to the integration bus. The message remains in the hospital during all re-tries until the subscribing adapter successfully processes it or the maximum allowed retries is reached. The subscribing application adapter contains two collaborations for each message family. One collaboration is triggered to process incoming messages (the "subscriber" collaboration) and the other (the "retry" collaboration) is dedicated to re-publishing messages in the Error Hospital back to the JMS queue. Every subscriber adapter has a unique "retry" event type, which allows some adapters to retry a particular message even if others have processed it successfully.



Subscription Process

Figure3-4

Figure 3.4 illustrates the processing involved for these messages:

1 The appropriate collaboration is triggered by a message from a JMS provider hosting the RIB Queue. This message may arrive on the RIB Queue from the Error Hospital, from a publishing adapter, or from a TAFR adapter.

2 The Error Hospital Java code is called to see if this message should immediately be placed into the Error Hospital. This logic will check

   a To see if any previously processed messages for the same business entity is in the hospital. If so, then this message needs to be put into the Error Hospital to preserve message sequencing.

   b If this is the second time this message was processed because the stored procedure returned an error the first time. If so, then the expectation is that the message needs to wait a while before it is retried. The message is placed into the Error Hospital to allow other messages to flow through during this time.

   If the message is placed into the Error Hospital in this step, the database work is committed and the message is removed from the RIB Queue. Steps 3-6 are not executed.

3 The correct Message Family Manager stored procedure is called. The specific stored procedure called is based on the message type of the message.

4 The stored procedure executes the appropriate application specific logic. This may involve direct updating of application logic or simply inserting the data into staging tables.

5 If step 4 returns an error, the message is flagged as “bad” (see step 2), and the transaction will be rolled back. The message is kept on the RIB Queue. The next time the message is processed, it will be put into the Error Hospital.

6 If step 4 returns success, the collaboration returns success: all database updates are committed and the message is removed from the RIB Queue.

At the end of each attempt to process a message, it is found in exactly one of three locations: Still on the RIB queue (because of a stored procedure problem), in the Error Hospital, or successfully consumed by the subscribing application.

Subscribing application stored procedure APIs

The concept of a Message Family Manager (MFM) is also used with message subscriptions within the RIB. However, instead of a single PL/SQL package processing all message types within a Message Family, a subscribing MFM uses a single PL/SQL package to process a single Message Type within a Message Family. As in the publishing side of processing, the subscribing MFM is only concerned with the XML Payload and not the entire RIB Message XML.

All MFM packages that parse and process the payload within a RIB message have the same procedure name (CONSUME) and same basic parameter list. An example is seen below:

```
PROCEDURE CONSUME(O_status_code          IN OUT  VARCHAR2,
                  O_error_message           OUT  VARCHAR2,
                  I_message              IN OUT  CLOB);
```

where

`O_status_code` is the success/failure status of the procedure call. The values of this parameter that are standard across all subscribing packages are found in the RIB_CODES package. Currently, these include:

`SUB_FATAL_ERROR` – A fatal error was encountered processing the payload.

`SUB_XML_PARSE_ERROR` – The payload could not be parsed due to a validation error.

`SUB_SUCCESS` – The payload was processed successfully

`O_status_code` may also contain values that are application specific. These values must not conflict with those listed above. These values should be listed in the *Retek 10.1 Integration Guide.*

`O_error_message` is text associated with any error condition.

`I_message` is the payload XML text used as input to the stored procedure.

Additional parameters may be present, depending of the specific MFM/Message Type that is processed.

Error Hospital

The Error Hospital is a set of Java Classes and database tables that are designed to segregate and trigger re-processing for messages that either:

- Had some error with their initial processing.

  or

- Update the same business entity as the one mentioned above.

Each time the message is re-processed, a record is kept of the event along with the results. The intent is to provide a means to halt processing for messages that cause errors while allowing continued processing for the "good" messages.

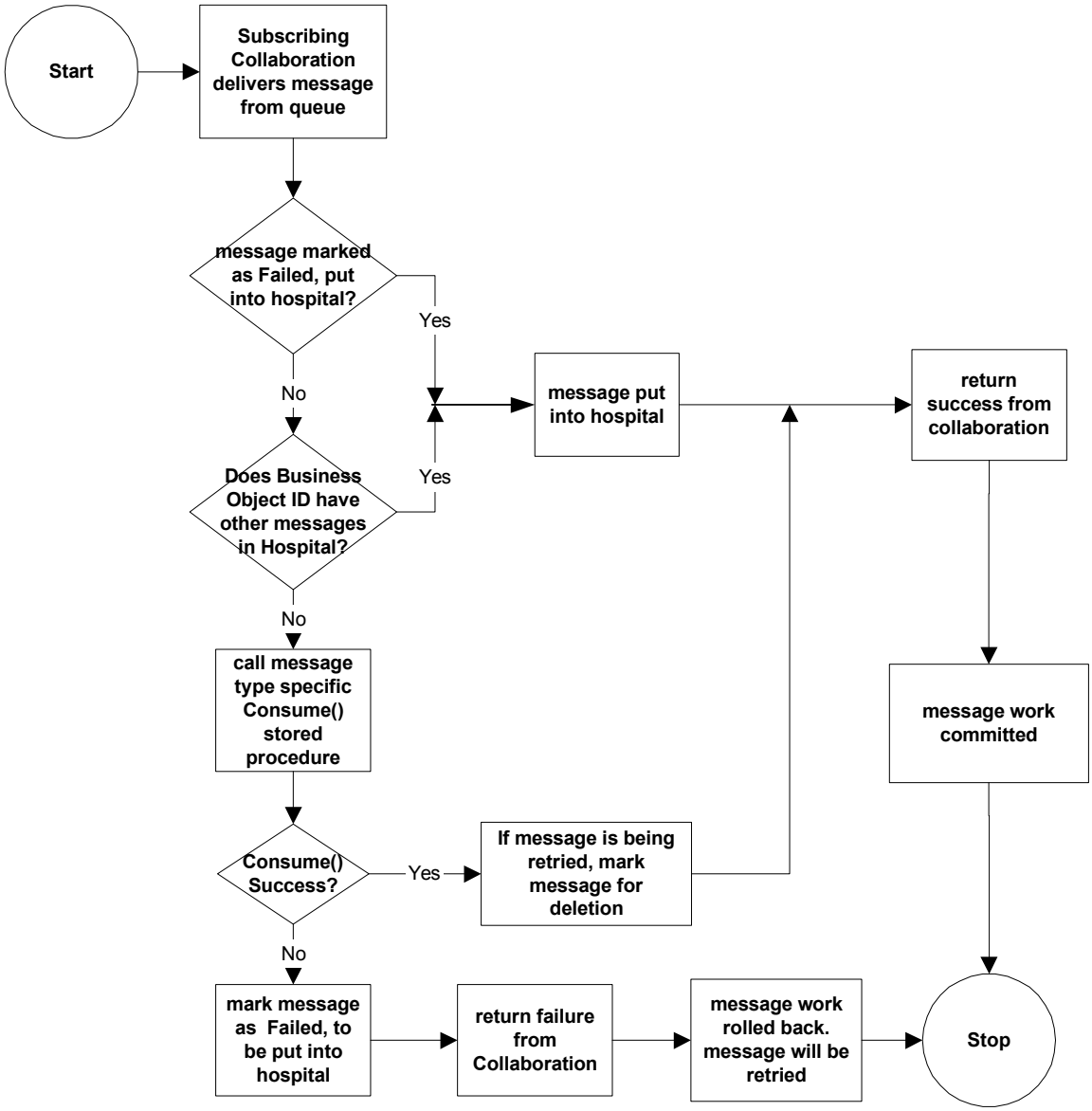
If a message is to be inserted into the Error Hospital because of an error during processing, it is sent to the subscribing collaboration twice. This is because subscribing collaborations are executed within the context of a distributed transaction, using the XA two-phase commit protocol. This transaction is controlled by the e*Way infrastructure: If the collaboration returns success, the message is removed and all database work committed. If the collaboration returns failure, the message never leaves the integration bus queue and the database work is rolled back.

Note: The XA interface is a standard protocol between a "Transaction Manager" and a database or "Resource Manager". In a SeeBeyond e*Way, the Transaction Manager is part of the e*Way software that is involved in executing the collaboration. Note that both the RIB queue and the database Connection Point must be configured to support the XA protocol. For more information regarding the XA standard, see the URL http://www.opengroup.org.

When the initial failure occurs while processing the message, the error is flagged within the Error Hospital software, the collaboration returns failure so that the database transaction is rolled back, and the message is kept on the integration bus queue. Because the message has not been successfully processed, it is re-submitted to the collaboration. This re-try will now cause the message to be inserted into the Error Hospital tables.

The Error Hospital assumes that each message family has a single unique ID for all entities its messages affect. This ID must be the same for the same entity across all message types within the message family. The reason for this is that the Error Hospital will automatically insert subsequent messages for the same entity to maintain the sequential order. Otherwise, multiple updates to this entity may be processed in the incorrect order, thus leaving incorrect values for that entity.

For a subscribing adapter, the following logic is performed regarding placing messages in the Error Hospital:



Subscriber Error Handling Logic

Figure 3-5

Also associated with the Error Hospital within the subscribing adapter is a "Retry" collaboration. This thread of control is responsible for retrying "sick" messages and to delete all messages marked for delete. It selects messages to retry based on the Business Object ID, the "Hospital ID" (a sequence number used to insure message sequencing is maintained), and whether the maximum number of automatic retries has been reached.

The following tables are used to store messages in the Error Hospital:

- `rib_message` – contains the message payload, all single-field envelope information, and a concatenated string made from <id> tags. Also contains a unique hospital ID identifying this record within the hospital.
- `rib_message_failure` – contains all failure information for each time the message was processed.
- `rib_message_routing` – contains all of the routing element information found in the message envelope.

Additionally, a sequence, rib_message_seq, is used to maintain a unique "Hospital ID" associated with each message placed into the Error Hospital.

Note: The "Retry" collaboration is responsible for maintaining the "State" information for hospital records. One element of this information is whether the message has been queued to the RIB Queue for re-try processing. Thus, manually deleting messages from the hospital database using SQL directly may produce severe processing problems. Similarly, deleting messages directly from the JMS provider may result in a message that is never retried again.

The RIB is supplied with a command-line and GUI interface to the Error Hospital database for administrative message control. These facilities also allow one to manually change the payload data for the next retry attempt.

Performance and "M of N" Threading

The default message-processing paradigm is to have a single publisher for all messages, zero or more TAFR adapters (depending on the subscribing system(s)) and one subscribing adapter per subscribing application. All of these components originally come supplied within a single RIB schema and are placed on a single host system. By implementing a production system using these defaults, one is limited to the processing capabilities of a single RIB host and associated database server. In some situations, this may not be sufficient.

There are a couple of ways one can deploy the RIB with higher throughput capabilities. The first is to offload a subset of the RIB components to another host. This is easily performed using the SeeBeyond Enterprise Manager. This is applicable when a single RIB server is 'maxed out' in terms of one or more resources. For example, if the RIB Server is CPU bound, or I/O bound. The determination of which component(s) should be based on the source of the problem. Such as:

- If the problem is Disk I/O on the RIB Server, then the problem may be due to a single JMS provider on the host and creating another JMS provider (JMS IQ Manager) on another host may provide the answer.
- If the problem is with CPU and/or network I/O on the RIB Server then one may consider moving some of the high-volume adapters to another host. Additional JMS providers may also need to be created.

But, the problem may be one of the serial nature of processing the messages. For example, the message family manager GETNXT() procedure may be taking too long to meet the message publication requirements. Assuming that the database tables are indexed correctly, the next step in achieving performance goals may be to multi-thread the processing from end-to-end.

Note: A huge increase in performance of an MFM may be achieved if the staging tables are indexed correctly. The difficultly for this lies in the fact that the near-real time nature of the RIB means that small numbers of records are usually present at any given time. However, batch processes may infrequently produce large amounts of data within a short period of time. Thus, when creating indexes for the staging tables, it is important to generate them when large amounts of data is present in the staging table. Otherwise, incorrect or ineffectual indexes or statistics may be used.

The current RIB architecture supports this for a set of publishing Message Family Managers. These, "High Volume" MFMs contain a 'GETNXT()' stored procedure that contains two parameters used for multi-threading the publication process: I_NUM_THREADS, and I_THREAD_VAL. I_NUM_THREADS is the total number of threads that are involved in the publication process. I_THREAD_VAL is the current thread 'number'. This has been termed "M of N" threading because each collaboration considers itself the Mth thread out of a total of N threads.

The values of these parameters are set in the rib.properties file. The properties of interest are:

mfm.<eway>.total_threads	I_NUM_THREADS value
mvm.<eway>.<collab>.thread_num	I_THREAD_VAL value (current thread number)

where:

<eway> is the name of the e*Way
<collab> is the name of the collaboration in the e*Way

An example entry for the RMS Purchase Order publisher (ewOrderPhysFromRMS) with two publishing collaborations is:

```
mfm.ewOrderPhysFromRMS.total_threads=2
mfm.ewOrderPhysFromRMS.ewOrderPhysFromRMS.thread_num=1
mfm.ewOrderPhysFromRMS.ewOrderPhysFromRMS.thread_num=2
```

The publishing collaborations may be deployed among multiple e*Ways, as long as:

1 No duplicate “thread_num” values exist.

2 The aggregate set of “thread_num” values contains all valid values (1 to “thread_num”, inclusive)

3 the total_threads property is the same for all of the e*Ways containing the collaborations

Once additional publishing collaborations have been created, one must determine if the downstream processing for the messages also must be duplicated. Doing so allows one to distribute the processing stream across multiple hosts. It also allows for multiple subscribers to process the same message family while at the same time insuring the message sequencing is maintained.

Chapter 4 – RIB message families

This chapter presents an overview of the RIB Message Families. Each Message Family contains information specific to a related set of operations. Processing by Message Family insures that a sequence of messages for a given Business Entity (for example, a PO) is maintained throughout the message lifecycle. In the RIB 10.1 release, a single thread of processing insures this sequence. The RIB infrastructure maintains a FIFO ordering for messages on all of its queues.

A Message Family may contain multiple "Message Types". Each message type encapsulates the information specific to a business entity within one or more business events. A single business event, such as updating a Purchase Order, may involve multiple business entities, such as a line item within the Purchase Order. Furthermore, because a single business event may involve multiple business entities, the application may publish messages for this event from multiple Message Families for a single business transaction. More than one message type within a Message Family may also be created.

Messages published from *different* Message Families do not have the same sequential guarantees as messages published from within the same Message Family. Examining the contents of the Message Families reveals that most, if not all, of these dependencies concern the existence of a specific business entity. Hence, an Item may need to be created before it is used in a Purchase Order. The Error Hospital retry logic alleviates this situation: when such a scenario occurs, the operation (say, creating a new PO line item detail) will be re-tried until the dependent entity (the Item) is created.

Event types and message families

Each Message Family uses a single SeeBeyond Event Type Definition to define the publishing format for all message types within the Message Family. Because of this, the SeeBeyond e*Gate Integrator infrastructure sees all messages from a Message Family as belonging to a single "type", known as the Event Type. The RIB message processing logic sub-divides the messages according to the message type field found in the RIB message envelope. The Event Type is the SeeBeyond ID associated with the type of the message. Event Types may use the same internal format. As such, Event Types may also be specific to how much processing has occurred on the data.

The SeeBeyond Event Type used for a Message Family may be changed if TAFR components are part of the processing stream. This is required when a single message needs to be routed to multiple destinations. In this case, each destination is associated with a distinct queue and each queue is associated with a distinct Event Type.

TAFR components may also change the Event Type messages when a mere transformation or filter operation is performed. This is done for two reasons:

1 It allows flexibility for the RIB topology. All messages may be put into the same queue on the integration bus if they have different types. For simple topologies, one can monitor the number of messages “In progress” on the RIB by looking at the statistics from a single queue.

2 It provides greater clarity when configuring a subscribing adapter or TAFR collaboration. Triggering events for a collaboration are fully specified by the Event Type and the source of the Event Type. When the source is an “upstream” collaboration, the Queue containing the event is “hidden” within the upstream collaboration’s configuration. Specifying the output event type using a different name insures that any components requiring the TAFR operation gets only TAFR processed messages.

Message family overview

This section is an overview of the Message Families contained in the RIB 10.1 release. Additional Message Families are expected in future releases. These message families are grouped by publishing application.

Remember that each RIB message is divided into an “envelope” and a “payload”. The envelope contains transformation, routing, filtering, retry, creation, and other information. The payload contains business event/business entity specific information. For more information on the RIB messages associated with each Message Family, see the *Retek 10.1 Integration Guide* manual.

RMS published message families

The following Message Families are published by the Retek Merchandizing System application:

Interface	Message Family Abbreviated Name	Description
ATP	ATP	Create or modify item/location stock data
Banner	Banner	Create, modify, or delete banner/channel information
Differentiator Groups	DiffGrp	Create, modify, or delete differentiator group headers or details.
Differentiators	Diffs	Create, modify, or delete differentiator information
Items	Items	Create, modify, or delete items and item/suppliers, item images and item UDA's
Locations	Stores	Create, modify, delete store information
Locations	WH	Create, modify, or delete warehouse information
Merchandise Hierarchy	MerchHeir	Merchandise hierarchy changes
Partners	Partners	Partner information
Purchase Order	Order	Create, modify, or delete a purchase order with physical or virtual warehouse items depending on the implementation of the RMS.
Purchase Order	OrderPhys	Create, modify, or delete a purchase order with only physical warehouse items
RTV Request	RTV Request	Return to Vendor Request
Seasons	Seasons	Season Information
Stock Order	Alloc	Create, modify, or delete allocation header and detail information
Stock Order	Transfers	Create, modify, or delete transfer header and detail information
UDAs	UDAs	Create, modify, or delete User Defined Attribute information
Vendor	Vendor	Create, modify, or delete suppliers and supplier address information
Work Order (Inbound)	WOIn	Create, modify, or delete inbound work order information. This includes both an entire work order, header modifications or detail modifications or deletions.

RDM published message families

The following Message Families are published by the Retek Distribution Management application:

Interface	Message Family Abbreviated Name	Description
ASN Inbound	ASNIn	Advanced Shipment Notice (inbound shipment) creation, modification, or deletion.
ASN Outbound (BOL)	ASNOut	Creation of Advanced Shipment Notice or outbound shipment from warehouse (Bill of Lading)
Customer Return	CustReturn	Customer return
Inventory Adjustments	InvAdjust	Inventory adjustment message
ItemWH	ItemWH	Items within a warehouse
Receipts	Receipts	Stock receipt creation and modification.
Receiving	Receiving	Appointment creation or deletion, header modifications and appointment detail create, modify, or deletion
RTV	RTV	Return to Vendor Transfer
Space Locations	SpaceLocs	Space location information for a warehouse
Stock Order Status	SOStatus	Status or modification of a stock order (Transfer or Allocation)

RCOM published message families

The following Message Families are published by the Retek Customer Order Management application:

Interface	Message Family Abbreviated Name	Description
Customer Back Order / reservation	COBoRes	Customer Back Order creation, change to/from reservation, cancellation
Customer Return Sale	COReturn	Customer return of a sales item.
Customer Sale	COSale	Customer sales
Locations	ShipMeth	Shipment Method notification creation, modification or deletion.
Pending Return	PendReturn	Pending customer return creation, deletion, detail creation, detail modification, or detail deletion.
Stock Order	CustOrder	Customer order

Externally published message families

The following Message Families are published by non-Retek applications.

Interface	Message Family Abbreviated Name	Description
Currency Rates	CurRate	Currency Rate information typically published by a Financials application adapter.
Freight Terms	FrtTerm	Freight Term information typically published by a Financials application.
Payment Terms	PayTerm	Payment Term information typically published by a Financials application adapter.
Locations	Locations	Location creation, modification, or deletion.
Vendor	Vendor	Vendor information typically published by a Financials application adapter.
Stock Order	StockOrder	Stock Order creation, deletion, header modification, detail creation, detail modification, and detail deletion.
GL Chart of Accounts	GLCOA	General ledger chart of accounts, typically published by a Financials application adapter.
SKU Optimization	SKUOptm	SKU Optimization tasks.

Chapter 5 – External application message interfaces

This chapter presents a brief overview of interfacing with external applications using defined RIB messages.

RIB message paradigm concerns

The following tenets of the RIB Messaging system are of interest to external (non-Retek) publishers and subscribers:

1 When a business entity is created, some result in one or more "Create" messages. These messages consist of all header and detail information for the composite entity created. External applications may require that these messages be coalesced into a single composite message.

2 Conversely, an external application may not have the same data model as the Retek application and require that a composite message be divided into multiple messages. These may need to be along the lines of a "header" and one or more "details".

3 When a business entity is modified, a message specific to the modification is published. The message may be specific to a sub-entity. For example, if a line item is added to a Purchase Order, a PODTLCre message will be published. If multiple items will be added, multiple PODTLCre messages will be created. This means that a single database transaction may result in multiple messages within the same or multiple message families.

   Non-Retek subscribing applications cannot associate a single message with a single database transaction.

   In terms of non-Retek publishing applications, the application must publish using Retek's canonical form (as specified in the Retek Integration Guide) or convert to this format. Besides converting field names or code values, this may also mean splitting up a single message into multiple messages.

4 Deletion messages may be applicable to an entire composite business entity. Different message types distinguish between the deletion of a sub-entity and the composite entity. For example, a Delete Supplier message will delete the supplier and all of its addresses, while a Delete Supplier Address will only delete a supplier's address.

   Non-Retek subscribers that cannot accept a single delete message for these entities will need to have additional processing to specify the sub-entities to delete.

5 The full create/modify/delete/detail update/detail modify/detail delete message types are *not* available for all message types. Non-composite business entities do not contain "detail" operations. Some messages, such as a Stock Order Status, reflect only an adjustment to an entity that will never be deleted (or created) by the publishing application.

   Non-Retek applications must only publish messages supported by the RIB if they are to be consumed by standard Retek applications.

RIB published messages may require modification or transformation to satisfy the external application APIs. These modifications and transformations may involve additional database operations. For example, the complete vendor name may be needed in a message as opposed to a "vendor ID" found in the RIB message. Once the data requirements of the subscriber have been determined, the available RIB messages should be inventoried for their applicability and the specific transformations that need to be applied to them.

SeeBeyond application-specific adapters

When integrating with an existing non-Retek application, development time may be shortened considerably using a SeeBeyond e*Gate Application Adapter designed for that specific application. These application adapters are either:

- e*Ways that surface an application's interface via a set of event type definitions: For these types of e*Ways, one must develop a set of subscribing collaborations that accept RIB messages as input events and a set of publishing collaboration that accept the application specific events.

  The subscribing collaborations convert the input RIB event into the event types associated with the non-Retek application adapter. Then the collaboration must publish the event to the "External" side of the e*Way. The "external" side then understands what API's are used for each event type and updates the application with the correct data.

  The publishing collaborations must convert the input application specific events into one or more RIB events before publishing them. The source of these events must be the "External" side of the e*Way.

  Because of deployment limitations and performance concerns, it may be necessary to locate the message event type transformation logic within a different e*Way or BOB from the application specific e*Way. Because the conversion is already done, no transformation is needed at the application specific e*Way and "pass-through" collaborations are configured as part of the e*Way.

- A library of event type definitions or wizards used to create these ETDs: An example of this is the EDI ETD library. The purpose of these libraries is to reduce the time creating, parsing, and/or validating the message format. For example, one could use the event type definitions for EDI. In this case, the ETD library aids parsing of the EDI document and reduces the amount of development needed to convert these into messages used on the RIB.

Chapter 6 – Retek Extract, Transform, and Load

The Retek Extract, Transform and Load (RETL) is a high-performance runtime tool that is especially useful in parallel processing systems designed for high volumes of data. The design of the RETL decreases the time importing or exporting data to or from a database. An “IMPORT” operation reads from a data file and an “EXPORT” operation creates a data file.

The usage of the RETL tool should be based on desired performance and data volume. The RETL is a tool that leverages parallel processing. Although the integration bus can also be configured for parallel processing, the RETL tool set is much more flexible, and performs better. RETL is optimized specifically for high data import and export throughput – much more than a normal on-line messaging system.

The RETL software is extremely powerful and flexible. There are currently no standard event type definitions for the RETL. The relationship between the RETL and the RIB integration bus intersect only on the transfer of these files. As such, one should treat the RETL tool in the same manner as a batch job stream. The RETL may use a file as input or create a file as output. These files may be transferred like a regular batch file. However, if the RETL is used between two Retek databases, it may make sense to keep the file where it was generated and to create two batch jobs executing serially on the same host.

Note that the size of the files produced could be a concern when RETL is used. As seen in the next chapter, the easiest way to implement a batch file transfer is as a single message. However, the one-to-one association of a file to a message requires that the entire message must be read into program memory. If the file is very large, then this could consume more resources than are available, causing the file transfer e*Way to hang or error. Hence, it may be worthwhile to investigate the size of the files imported or exported via the RETL tool and, if over 100 megabytes in size, consider techniques to break the file up into smaller sizes.

Chapter 7 – Batch job integration

The main characteristic of a batch job is the reliance on a file as the means for input and output. In point-to-point solutions, this file is typically FTPd between systems. To integrate with the RIB, the batch file is converted to one or multiple messages published to the integration bus.

The does not exist any pre-packaged batch integration software within the RIB 10.1 software that extracts data from the database and publishes it as a series of RIB messages versus a file. If such software existed, then this in itself would be a message-based solution (and there would still not be any pre-packaged “batch” integration). However, the SeeBeyond e*Gate Integrator infrastructure allows files to be used as sources or sinks for messages.

The RIB may be an alternative to using FTP or in conjunction with FTP file transfers. The mechanism currently used to FTP existing batch jobs may be replaced completely RIB based mechanisms.

Motivations for replacing FTP transfers

FTP is a common method for transferring files between systems. It uses a stable, well-specified protocol and mature products are available that implement it. RIB integration with batch files involves taking the file information and publishing it to the RIB. The reasons why one would want to replace an FTP transfer with this method include:

- Reduced number of FTP jobs that transfer the same file from place to place.
- With FTP, both hosts need to be available. When an adapter publishes data to a RIB queue, only the RIB and one of the hosts need to be available. Because of the distributed processing available on the RIB and the ability to move components physically within a network, there is an increased flexibility for operations personnel to perform system maintenance.
- Subscribers or publishers can move from a batch-oriented method to a message-oriented mode in an incremental fashion. After publication, file data exists as one or more messages and can be transformed, filtered, and routed as such. If the same data is needed by multiple subscribing applications, then some of the subscribers can remain relatively unchanged and still use a file as input while others can read the data as messages directly from an integration bus queue.

Transfer file data using a batch application e*Way

The first and simplest available option for using the RIB in this respect is to use the SeeBeyond e*Gate Batch application e*Way to transfer file information to and from the RIB. This e*Way can be used to copy files to or from hosts without installed e*Gate components. The Batch e*Way is fully documented in the SeeBeyond *Batch e*Way Intelligent Adapter User's Guide***.** This manual presents a brief overview of its capabilities.

Do *not* use the SeeBeyond e*Gate File e*way. This is a development tool not robust enough for deployment in a production environment.

A batch e*Way is created by creating new e*Way in the e*Gate Enterprise Manager, selecting "stcewgenericmonk.exe" as the "Executable file", and then, when creating the new configuration file, selecting the "batch" e*Way configuration template.

The Batch e*Way works in one of two modes:

1. A *fixed* configuration that publishes data to the RIB based on the presence of a file in a directory or creates/appends a file based on the presence of a message on a queue.
2. A *message* based configuration where the batch e*Way subscribes to messages that contain the specifics of the file transfer.

"Fixed" configuration

Publication of data to the RIB

A batch e*Way is configured to poll for the existence of files (either on the local system or on a remote system). Once found, the e*Way copies the files to a local temporary directory. For files found on remote systems, FTP is used to copy it to the local temporary directory. Configuration options determine the polling interval, where the file is located, file masks to determine which files to transfer, FTP parameters, whether the file should be renamed or archived after publication, and if the contents of the file should be published as a single message or if each line in the file corresponds to a single message. This is all performed in the "application" side of the e*Way.

Once a message is ready on the application side of the e*Way, the message is sent to the "collaborations" configured with the e*Way. A collaboration must be created that can handle the messages published whose source is "<external>". In the simplest case, this collaboration could merely pass through the data without modification or validation. In a more complex case, the collaboration could validate and transform the data before publishing it as an event.

If the entire file is to be published as a single message, the entire file will be read into the memory of the batch e*Way. The memory allocated for this may never be relinquished by the e*Way, depending on its scheduling. Severe problems may result when the amount of memory needed exceeds the maximum available for a single process or when the virtual memory of the machine is exhausted. Retek internal test systems successfully transferred files 100 megabytes large; your results may vary according to the specific operating system and its configuration.

Subscribing to data from the RIB

A batch e*Way is configured with a collaboration that is triggered from events (messages) published by another collaboration or are available on a JMS queue. The processing order of these events is the reverse of publication. First, the subscribing collaboration is executed and performs any needed transformations or validations. Then the message is passed over to the "application" side of the e*Way by publishing the message to the "<external>" destination.

The configuration of the application side determines the final disposition of the data. As in the publication scenario, the data stages through a temporary file and before copied to its final destination. FTP is used when the final destination is a remote system. Configuration options for this processing include the following:

- The name of the file to put the message in.
- Whether messages are appended to this file or new files are created.
- Whether the file is uniquely named via a time stamp or sequence number.
- How often new files are created (if the append mode is used) and copied.
- Pre- and post- file copy activities.
- FTP session parameters.

Import notes: When the "append messages to a file" is used, file boundaries are not necessarily maintained from the source file. One or more source files could be put into a single destination file or, if the source file was published record-by-record, half of the source file could be appended to a single destination file and half to the next. It all depends on a set of interacting configuration parameters. Furthermore, if a batch e*Way was used to publish the file using a "fixed" configuration, no intrinsic mechanism exists for communicating the name of the source file.

“Message” mode

In message mode, the batch e*Way receives an XML message detailing the file transfer details. This message contains one or more operations or commands to execute. There are two types of commands:

1 “receive” – find one or more external files and publish them to the integration bus. The message published by the e*Way is formatted using XML. It contains an identifying “return_tag” plus a “payload” tag containing the data found in the file.

2 “send” – the subscribed message is used to create or append to a destination file. The message contains a “payload” tag with the file contents. Other tags in the message detail other specifics of the file, such as the destination file name, and what to do if the destination file exists, and local/remote file copy details.

One advantage of the “message mode” FTP configuration is that “send” commands specify the name of the destination file. Hence, it is possible to maintain file names across the file transfer. However, this method requires additional development and processing.

Transferring data directly from/to a database

Another method for implementing batch transfers is to create an e*Way and a set of collaborations to read from a database table and publish the information to the RIB. This involves using the e*Gate Enterprise Manager to create the event type definitions, collaboration rules, collaborations, e*Ways and queues. This strategy replaces a batch mode of processing with a message-based mode. It directly uses new development specifically for the integration bus.

There are two strategies one can use for this development: Using connection points and developing the logic entirely within a collaboration or using one of the “Generic” SeeBeyond e*Way adapters.

Using connection points and developing the logic within a collaboration

This strategy is useful if the data is available via a simple SQL statement or with little added processing. (Actually, the wizard generates events based on table structure, SQL statement, or Stored Procedure API.) The e*Gate Enterprise Manager contains a database wizard that can generate an event corresponding to the SQL statement.

Publication: One defines an e*Way connection with a polling parameter determining how often these events will trigger the collaboration. No data or SQL statement will populate the event (message) when the collaboration triggers. The SQL statement executes as part of the collaboration rule logic and each row of any result set needs publishing as a separate event.

Subscription: One configures a collaboration that includes the defined event as an output event with a destination specified as a database connection point. The collaboration transforms the input data into the SQL specific event and then executes the SQL statement.

Note that database transaction boundaries depend on XA interface usage and an event's destination or source. If the XA interface is used, all work within each invocation of the collaboration is within a single transaction. If not, the collaboration can execute multiple transactions per single invocation. RIB collaborations typically use XA to insure "exactly once" successful message processing.

Using a "generic" e*Way application adapter

A Generic e*Way Application Adapter is useful when the business logic surrounding message creation or processing is not trivial. This series of adapters also *cannot* leverage the XA interface. There is the possibility that the same message is published or consumed multiple times.

Generic Application Adapters are specific to a programming language such as Java or C/C++. Their configuration specifies a shared library, DLL, or Jar file that contains the application logic. The functions, classes, and methods used in this logic must meet certain criteria.

These adapters have the following models:

- **Publication:** When the e*Way is instantiated (brought up) its configuration is read and the container of the application logic is attached to the e*Way. Specific initialization functions are called (as per the Generic e*Way standard application API). These functions may perform one-time activities, such as establishing a database connection. Additional functions or methods need to be implemented to inform the e*Way of lost connections or other events. Once the e*Way is initialized, it polls (according to a configured parameter) the application by calling a specific application provided function. If any data is available, the e*Way attempts to decode the returned bytes as a message in order to invoke a collaboration to process this message. All collaborations of this sort must subscribe to an event whose source is "<external>".

  The collaboration may simply pass the message through for publishing as-is or transform the event in some way. Once the message has been published successfully, a function is called on the "application" side of the e*Way to allow the application to further update state or commit updates already performed. The application polling function is called again and the process repeated. When the collaboration processing the application's message returns failure, the e*Way calls a "failure" function to allow the application to process the failure or rollback database changes.

  Between each loop there are checks to see if any the application is ready to continue or if an administrator has requested the e*Way to shut down.

- **Subscription:** In order to process incoming messages, a Generic e*Way must have at least one collaboration configured with an output event type that the application can parse. This event must also have a destination of "<external>". Input events can come from any valid connection point or other collaboration. The collaboration processes the input event according to its own logic and publishes the output event. The e*Way presents the output event (message) as a parameter to an application-side implemented function.

Note that the application side of the e*Way is responsible for maintaining its own database connections that it uses. Any needed information can be prompted for in the e*Way configuration using modified “configuration definition files” (*.def).

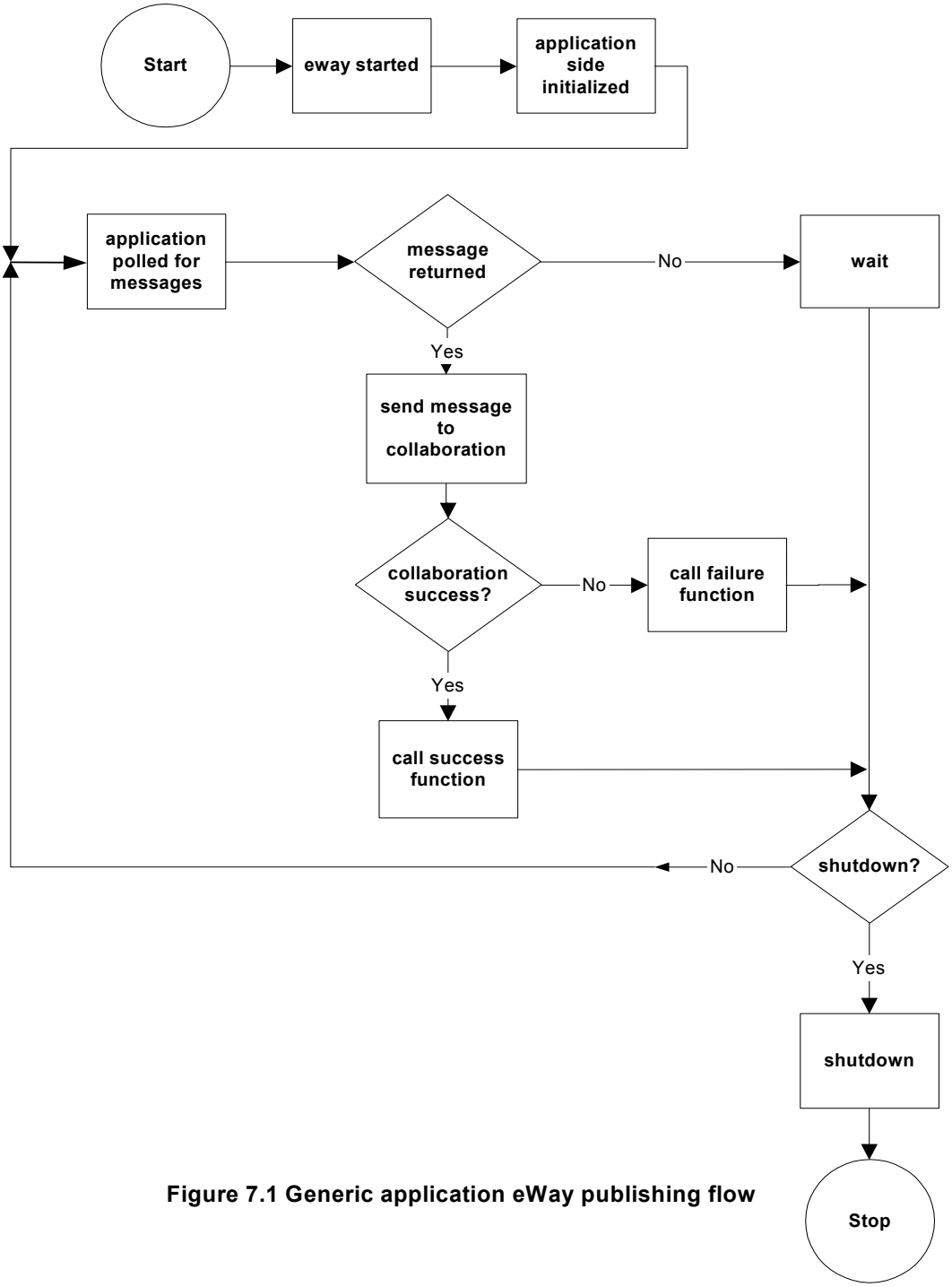


Figure 7.1 Generic application eWay publishing flow

For more information on the specifics of the Generic e*Way adapters, see the appropriate SeeBeyond manual listed below:

- Java Generic Extension Kit Developer's Guide
- C Generic e*Way Extension Kit Developer's Guide
- Generic e*Way Extension Kit (Monk enabled)

Using an application specific e*Way adapter

Application specific e*Way Adapters are built using the same paradigm as the "Generic" adapters listed above. However, these e*Ways have the "application side" of the e*Way already developed. The event types (message formats) the application can publish or parse are typically defined already (or at least an easy way to create them is available) along with the application processing logic. Hence, the main work here is to develop the correct collaborations to convert RIB events (messages) to or from this set.

There is a rich set of application specific adapters available. A complete list is available on the SeeBeyond web site, http://www.seebeyond.com.