Retek® Integration Bus™ 10.3.4

Operations Guide

Corporate Headquarters:

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

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

Fax:
+1 612 587 5100

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 email, 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. Retek customers on active maintenance agreements may contact a global Customer Support representative in accordance with contract terms in one of the following ways.

Contact Method	Contact Information
E-mail	support@retek.com
Internet (ROCS)	rocs.retek.com Retek's secure client Web site to update and view issues
Phone	+1 612 587 5800

Toll free alternatives are also available in various regions of the world:

Australia	+1 800 555 923 (AU-Telstra) or +1 800 000 562 (AU-Optus)
France	0800 90 91 66
United Kingdom	0800 917 2863
United States	+1 800 61 RETEK or 800 617 3835
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 – RIB component overview

Introduction

This manual is designed for System Administrators, Developers, and Applications Support personnel. Its purpose is to provide a basic understanding of the Retek Integration Bus components, how messages flow between them, and operational activities surrounding these components. It also provides templates for using the RIB as an alternative to FTP batch jobs for transferring files from one system to another.

This chapter describes the components that make up the Retek Integration Bus (RIB). These components are distributed within the SeeBeyond Technology Corporation's (SeeBeyond) e*Gate™ Enterprise Application Integration platform. The final deployed system may be distributed across multiple computing systems. These systems may be running a Microsoft Windows, Unix, or Linux operating system.

If the SIM/ISO module has been purchased, Retek's ISO application server (also known as the Chelsea application server) will be included with the actual SIM/ISO product. The RIB will then include some components that will be deployed into the ISO application server.

SeeBeyond components

Active messaging

This section contains a brief description of SeeBeyond e*Gate components. For more detailed information, see the e*Gate Integrator System Administration and Operations Guide.

In SeeBeyond's EAI environment, a "Registry" embodies a complete administrative domain. A Registry is a database defining the deployed EAI system and a program that controls access to this database. A Registry is organized into one or more *Schemas*. Each schema details a collection of *e*Ways*, *BOBs*, *Intelligent Queue Managers*, *Intelligent Queues*, *Connection Points*, and *Collaboration Brokers* along with their network addresses or locations. The Registry also contains basic security objects that control user identifications, roles, and privileges shared across all schemas.

Because the Registry embodies all configurable parameters, no other component can be brought up without access to a registry, either directly or indirectly. However, in a distributed environment, reliance on a single Registry can be problematic, since:

- System crashes or scheduled maintenance may bring down the Registry.
- Network partitions may occur that cut communication links between deployed components
- Reliance on a single host may produce a performance bottleneck.

Deploying and configuring "Secondary Registries" can alleviate these problems. Secondary Registries replicate the Primary Registry. The number and location of these Secondary Registries are dependent on the site-specific needs and capabilities of a deployed system. The replication of the configurations occurs transparently during normal operation of the system.

Each Registry is broken up into one or more *Schemas*. Each schema is a self-contained set of components that define "end-to-end" processing of one or more messages. The Schema contains the message processing units to deploy, where messages are stored, security roles, database access definitions, and other information. Schemas may be bridged, such that one schema may publish a message and other schemas contain one or more of the message's subscribers. For reasons of performance and high availability, schema contents can be copied within a single Registry (that is, two or more schemas are defined with the same component types and processing, but have different names and physical deployments).

In SeeBeyond's vocabulary, there are three types of logical computing host types: A Registry Host containing the Registry, Monitor Hosts where the e*Gate Monitor Software can be run, and "Participating Hosts" that produce, consume and process messages.

Note: This must be a Microsoft Windows NT/2000 platform. The complete requirements for such a system is detailed in SeeBeyond's e*Gate Integrator Installation Guide.

Although all three of these component types could run on a single physical host, this is rarely seen in production environments. Usually multiple computers are found in a deployed system – Operations personnel with PC's running the e*Gate.

All components within a Schema are defined within one or more Participating Hosts. There is a correspondence between a logical Participating Host and another SeeBeyond infrastructure component known as a Control Broker. The Control Broker is a program that controls the administrative activities for a participating host's messaging components (e*Ways, IQ Managers, and BOBs). The Control Broker maintains a network Connection with the Registry or a Secondary Registry at all times, because it also propagates configuration changes.

There must be at least one control broker up and running on any physical host involved in the deployed system. Furthermore, there may be multiple control brokers running on a single physical host because:

- The same computer may be configured as different "Participating Hosts" within a schema found in multiple Registries. This is because the same physical host may have multiple identifications within a Domain Name Server.
- The same host may be configured within multiple Schemas that are part of the same Registry.
- The same physical computer may be configured to hold multiple "Participating Hosts" within a single Schema.
- Any or all of the above may be true.

Each Control Broker starts with parameters detailing its own name and its associated Schema and Registry. At least one of these parameters must differ for each Control Broker instance. (That is, no two control brokers can start with the same name, same schema specification, and same Registry specification.)

Once a message is created, it usually needs copying to stable storage so that it doesn't get lost. The RIB uses the SeeBeyond JMS Intelligent Queue (IQ) Manager component for this. The JMS IQ Manager is a Java Message Service provider. Queues within the JMS system are identified as "topics" that publishers publish to and subscribers subscribe to.

Event types categorize the format of a message. The JMS IQ Manager equates an event type with a JMS topic.

The Retek Integration Bus uses the JMS IQ Manager extensively because it offers a two-phase commit capability. Two phase commits are integral to "exactly once" message processing.

Note: "Exactly once message processing" is a SeeBeyond product attribute that guarantees a message is processed only once successfully. This is important for non-idem potent messages – messages that contain "relative" values – that would cause discrepancies if processed by a subscriber more than once. For example, if a message reserving a stock item for a specific store could end up reserving all items for that store if processed enough time, even though the publisher only wanted one item.

The other SeeBeyond component deployed within a Participating Host is the *e*Way*. These components produce, consume, or otherwise process messages. This manual uses the term *adapter* as a synonym for an e*Way. All RIB adapters are e*Ways.

Besides the "application" side of an e*Way, messages can be produced or consumed from an entity known as a *Connection Point*. A Connection Point defines a session with an external entity such as a database, e-mail server, World Wide Web (HTTP/HTTPS) server, or Java Message Service provider. It is possible to poll Connection Point sessions for incoming data at regular intervals, as defined by their configuration. Multiple adapters may use the same Connection Point. Connection Point APIs may be multi-threaded and, depending on their design and configuration, support an XA compliant two phase commit. It is only through the XA interface that SeeBeyond insures a message is delivered and successfully processed exactly once.

The processing for a specific message used by an adapter is defined within Collaboration**.** The source of the message (or event) that triggers the collaboration's processing may be from either the e*Way application interface, from a Connection Point or from another collaboration. Messages published from collaboration must have an associated destination. This destination may be either an Intelligent Queue or a Connection Point.

One may use a Connection Point to ensure all processing performed on the message is done atomically. Connection Points implementing the XA interface can have a distributed transaction that enforces atomic commits and rollbacks. The e*Way's collaboration control logic manages the commitment or rollback of this distributed transaction based on the success or failure of the message processing within the collaboration. "Exactly once message delivery" requires the XA protocol and its associated two-phase commit operation. However, if the Connection Point does NOT implement the XA interface, then, under certain failure scenarios, the same message may be submitted for processing multiple times.

RIB collaborations will also fail if their database connection points do not support the XA protocol. RIB collaboration logic does not contain commitment or rollbacks. The distributed transaction must include the work involved in delivering the message from a queue to the collaboration. The collaboration starts only after the message delivery to the adapter. If an invalid connection point is used, then no database work performed by the collaboration logic will ever be committed.

The typical lifecycle of a message is as follows:

- First, the publishing adapter creates the message. The event that triggers the message creation may be a polling operation on the database, the presence of a file, or merely that a certain time interval has been reached. Each message is created in the context of collaboration, and part of the collaboration's configuration specifies where to publish the created message. The message is sent to a "queue" that then writes the message to stable storage.
- The message is now available to its subscribers. Subscription is based on the publishing collaboration / event type combination. Each subscriber will contact the queue and retrieve the next message available. Separate threads in the subscriber are used to retrieve messages on a per event type basis. The specific message retrieved from the queue depends on its location within the queue. As part of the retrieval processes, the Error Hospital software updates the state of the message to reflect that one of the subscribers is now processing it.
- Once a subscriber gets the message, it is free to process it according to its own rules. In the case of a transformer adapter, the subscribing collaboration can open the message, modify its contents, and then publish the modified message to a new queue. If the new message is of a different type than the original, the new message can be published to the original queue. There may be new subscribers to the modified message, and the scenario repeated for each of these subscribers.
- When each subscriber has finished processing a message, the queue updates the state of the message to reflect this. When all subscribers have finished with the message the message may be deleted immediately or be archived/journal led for a specific time before deletion. The archiving/journaling is specific to the type of the queue in use and the configuration of the queue manager.
- The JMS Queue Manager will delete the messages on the queue after delivering it to the appropriate subscribers or after it has been on the queue the number of seconds specified in the `MaxTimeToLive` configuration parameter.

Monitoring

So far, all of the components mentioned are actively involved directly in the EAI messaging system. In a production system, however, there must be a way to monitor the running system components.

Note: Monitoring the associated business processes occurs at a different level and is outside the scope of this discussion.

Four SeeBeyond components are useful in this respect:

1 The e*Gate Monitor: This application that allows an administrator to determine if a component is up or down and is responding to status requests. It also allows the administrator to bring up or down any component deployed on a participating host other than a control broker. Finally, it allows an administrator to interactively view and mark as resolved any e*Gate Alert Notifications.

2 The e*Gate JMS Administrator: This application allows an administrator to monitor the JMS Queue(s). JMS Topic and message statistics can be analyzed as well as the ability to view, edit or delete message currently in the queue.

3 The e*Gate Enterprise Manager: This application develops schemas or modifies existing schemas. As such, it is a primary tool for RIB development to create new Connection Points, e*Ways, BOBs, IQ's IQ Managers, Participating Hosts, user IDs, roles, etc., for a schema. A system administrator would also use this tool to modify the operational characteristics of schema components, such as changing the level of logging within an IQ or e*Way, the automatic running of e*Ways or BOBs, or specific database log-ins used in Connection Points. Unfortunately, these attributes may be changed when importing updated schemas from a test environment to a production environment.

4 Alert Agents or Monitors: Notifications of operational events, such as e*Ways going down, are passed from a control broker to one or more alert agents. Different types of alert agents exist and may be configured to create e-mails, console messages, and SNMP traps. The control broker creates notification events (messages) that these agents can process. See the following SeeBeyond manuals for more information on how to install, configure and modify system monitors:

   - e*Gate Integrator Alert and Log File Reference Guide
   - e*Gate Integrator Alert User's Guide
   - e*Gate Integrator SNMP Agent User's Guide
   - e*Gate Integrator System Administrator and Operations Guide

Retek supplied components

This section contains a brief description of how Retek has built upon the SeeBeyond platform to create the Retek Integration Bus.

The following components comprise the RIB:

- Database triggers that capture application activities as they occur. These triggers are part of the specific Retek application, such as RMS. However, as part of the RIB installation and configuration, they must be enabled to capture information regarding events of interest.

- Staging tables used to hold the captured information and to maintain the publishing state of the messages.
- Publishing e*Ways that create messages from the information captured by the aforementioned Database Triggers. These publishing e*Ways are designed to publish events from a single "Message Family" and are specific to a Retek Application, such as RMS. Each RIB publishing e*Way has a collaboration that will invoke a specific stored procedure which returns the staging table information.
- Subscribing e*Ways that are used to consume messages. These are specific to Retek Applications (RMS, RCOM, RDM) and are designed to consume all messages from a specific message family. Each Subscribing e*Way will call a specific stored procedure used to process a specific application event message.
- Transformation Address Filters/Router (TAFR) e*Ways that transform message data and/or route messages. The TAFR acronym is a generic term. Multiple, message family specific TAFRs have been implemented. Different TAFR e*Ways may be active on different message families or on the same message family depending on the needs of an application. Not all message families require TAFRs.
- Error Hospital database tables used as a basis for storing and re-trying problematic messages.
- Error Hospital administration GUI and command line utilities.
- Pre-defined Connection Points used by the adapters listed above. These must be configured after installation so that the correct database instance and logins are used.
- SeeBeyond Java Message Service (JMS) Queue managers. The JMS Queue Managers control the JMS queues used to store messages after publication. The messages persist on stable storage until all subscribers have processed them.
- For J2EE applications (RCOM, MDM, ISO, …), Enterprise Java Beans (Message-Driven and Stateless Session).
- If the SIM/ISO module has been purchased, ISO messaging components, and publishing utilities have been included for subscribing to RIB messages within ISO, and publishing RIB messages out of ISO. These components will act like e*Ways. Though they are developed under the ISO platform, they will still use the SeeBeyond JMS queue manager. They will subscribe to messages published by SeeBeyond e*Ways, and publish messages to the SeeBeyond JMS queue, to be consumed by subscribing e*Ways.

Additional resources

Use the following resources to further understand the Retek Integration Bus and the SeeBeyond e*Gate Integrator EAI platform:

- e*Gate Integrator Alert and Log File Reference Guide
- e*Gate Integrator Alert User's Guide
- e*Gate Integrator SNMP Agent User's Guide

  The three manuals above detail the options, configuration, and other reference material for creating Agents and other monitors for a deployed system.
- e*Gate Integrator System Administrator and Operations Guide

  Contains reference, troubleshooting and administrative information.
- e*Gate Integrator Installation Guide

  Contains basic information on how to install the SeeBeyond e*Gate Integrator platform.
- e*Gate Integrator Release Notes

  Useful if currently using an earlier version of the SeeBeyond platform.
- e*Gate Integrator User's Guide
- e*Gate Integrator Intelligent Queue Services Reference Guide

  Overview of the Intelligent Queues
- SeeBeyond eBusiness Integration Suite Deployment Guide

  This manual contains information on how to analyze, plan, and manage a RIB deployment.
- SeeBeyond eBusiness Integration Suite Primer

  This manual contains an introduction to all of the available components within the SeeBeyond e*Gate product family. These include e*Ways designed to interface to specific application suites, such as PeopleSoft, SAP, and Oracle Financials.

Chapter 2 – RIB component operations

This section details the message flows for a simple message and for a message undergoing a routing or filtering operation. For a more detailed description of the RIB components, see the *Retek Integration Bus Technical Architecture.* For a detailed discussion of message contents, see the *Retek 10.3 Integration Guide.*

Simple message flow

The figure below is a generalized view of a RIB message. Two applications require this data and subscribe to it. One subscribing application requires certain transformations applied to the data, but the other subscriber can process the message without any transformations.

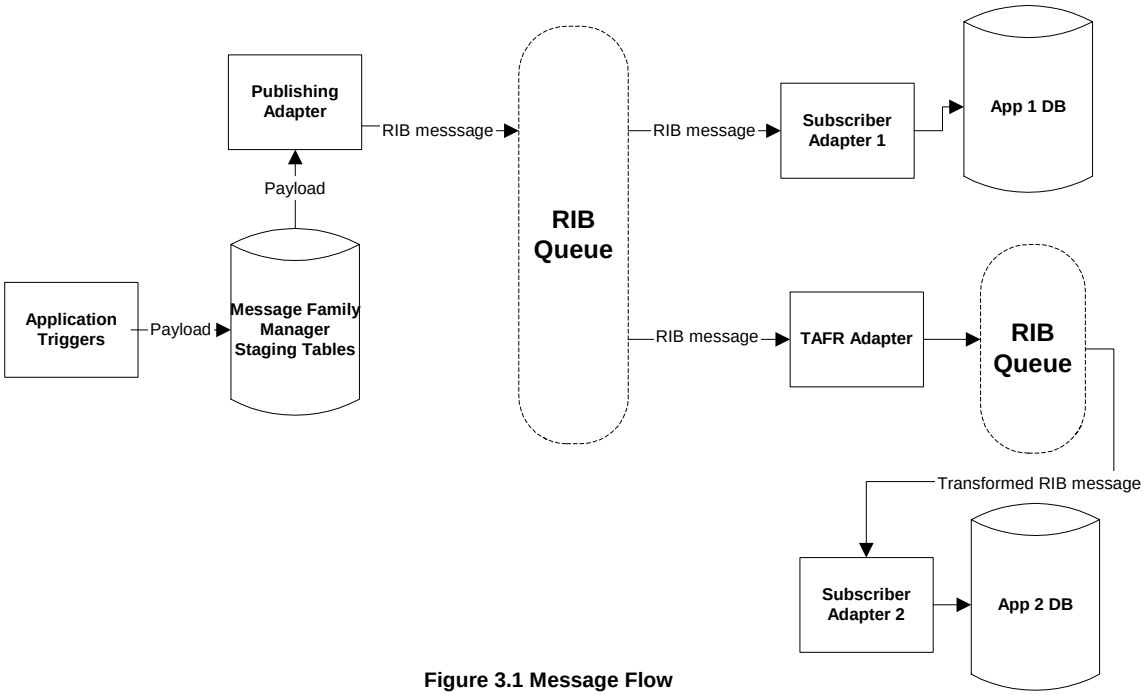


Figure 3.1 Message Flow

First, a trigger on a database table fires in response to an application's action.

Note: Some applications, such as RCOM, do not use triggers to publish to the MFM staging table. RDM uses another variation: an MFM interface harvests data from "Upload" tables to create the XML payload.

This trigger creates a row in a *Message Family Manager* (MFM) staging table and commits this data, known as the *payload*, along with all of the other changes performed by the user or batch job.

Second, a RIB Publishing e*Way polls the MFM staging table via a call to an MFM specific stored procedure. This stored procedure insures that messages are published to the RIB in the correct order and at the correct time. The Publishing adapter takes the payload and wrappers it with an *envelope* used by the RIB infrastructure. The publishing adapter then deposits the message on a Java Message Service (JMS) queue, which includes writing the message to stable storage.

Third, a RIB subscribing e*Way polls the JMS queue for a message and retrieves the one just published. Assume for simplicity's sake that this e*Way interfaces with the application requiring no data transformations. The e*Way then reads the data, performs any needed database updates, and commits all of its work. It is now ready to process the next message from the JMS queue.

Fourth, a RIB TAFR e*Way also polls the JMS queue. It retrieves the message, transforms it into a new message, and publishes it – effectively publishing a new type of message. The TAFR e*Way could publish the message to the same JMS queue it retrieved the message from using a different JMS topic or it can publish the message to a completely different JMS queue. The name of the JMS topic associated with the message may be determined from the message's Event Type name.

Fifth, the e*Way associated with the second application polls the second JMS queue, retrieves the message, and processes the transformed data.

Message routing

When a message requires routing, a TAFR adapter is needed that directs the message to the correct destination. The information it uses for routing is found within the message. However, the routing logic is tailored according to the needs of the subscriber.

TAFR routing logic many times consists of a simple chain of "if-then-else if" statements.

For example: *if* the routing tag equals "Warehouse1", *then* publish the message as event type "etMessageWH1", *else if* the routing tag equals "Warehouse2", *then* publish the message as event type "etMessageWH2", *else if* ….

However, the routing logic can be complex or route the same message data to multiple destinations. The determination of this logic is specific to the message family the TAFR is designed to process.

Once the message is published by the routing TAFR, it resides on a destination specific queue/topic combination. The TAFR collaboration configuration determines the specific queue used. There must be an association of the output event type to this queue.

From here, additional adapters retrieve the message and continue to process it. The logical flow diagram of a routed message as it travels on the RIB is seen in Figure 3.2. Note that the triggers and databases have been omitted from this diagram. Moreover, subscribers may publish additional messages, depending on the needs of the system.

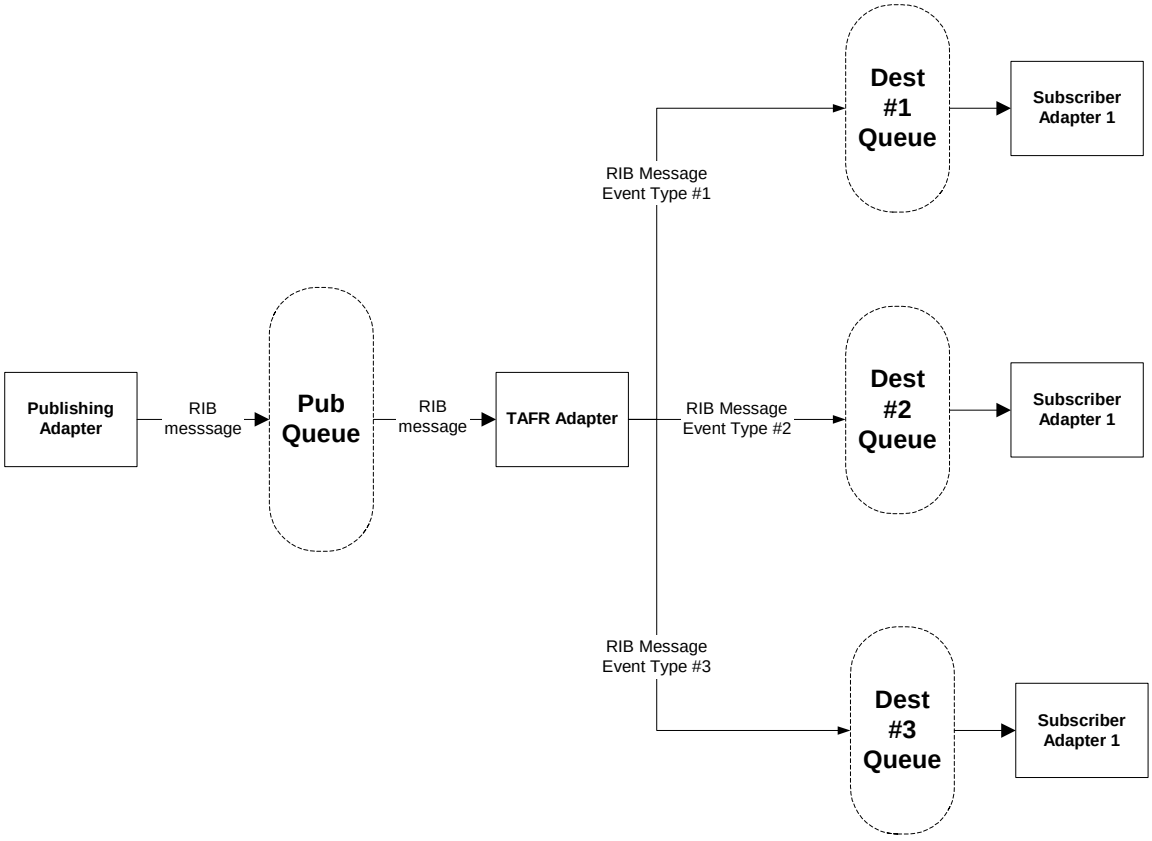


Figure 3.2 Routed Message Flow

Component failures

Understanding how messages are transported and processed successfully is a concern in a production system. An effective administrator needs to know what kinds of failure scenarios exist and what steps can be taken once these failures appear.

Application trigger failures

Failures involving the application database triggers should be extremely rare. When they occur, they manifest themselves as failures within the application. Trigger failures should be handled immediately.

Many triggers involve the use of a sequence generator as a primary key in a Message Family Manager staging table. If this sequence generator has been reset, then unique constraint exceptions may occur.

Another possible trigger failure also involves the insert operation into the MFM staging table: out of table space. As mentioned below, an analysis of the needed space should occur before deploying the system to production – or at least monitored closely while the system is in production. Messages must be published to the RIB before they are deleted from the staging table and if the publishing e*Way cannot keep up, the number of rows in this table and the publishing delay may increase to unsatisfactory levels.

SeeBeyond Publishing adapter failures

Failures involving SeeBeyond deployed publishing adapters (or e*Ways) may occur due to configuration errors or environmental errors. If a publishing e*Way becomes unavailable, then records will accumulate in the MFM staging table.

Configuration failures for publishing adapters may occur in the specification of its collaborations. Specifically, the configuration supplied as part of the initial product specifies an Oracle Database Connection Point used to trigger message publication. This Connection Point must have the correct database user login and SID information supplied or it will not work or a Connection Point must be specified that contains the correct information.

Similarly, publishing adapters specify a JMS Connection Point for the JMS queue the message is published to. If a SeeBeyond JMS queue is used, then the JMS Queue Manager must be set up and attached to the Connection Point. Otherwise, all messages will fail when published.

Another common problem with publishing adapters, or any adapter, is that RIB collaboration rules (the processing logic) are written in Java, and the correct CLASSPATH must be specified in the environment or in the e*Way's configuration. If one uses all default file directory locations, it is expected that this variable will require little or no modifications. However, if the SeeBeyond e*Gate system or the Java Runtime Environment is installed in an unexpected location, then all RIB publisher, TAFR, and subscriber adapter configurations may need to be modified.

Similar to the CLASSPATH problem, but more insidious, is the JNI DLL specification.

Note: The term "DLL" is used even on Unix systems within the e*Gate product. This is even though DLL's are specific to a Microsoft platform. On the Unix platform this refers to the JNI shared library.

This is the Java Native Interface used within an e*Gate e*Way to jump from a Java context to native C or C++ context. The JNI DLL specification specifies where the library containing the "jump" code is located. It is considered part of the run-time environment.

SeeBeyond deployed TAFR adapter failures

TAFR adapters use collaborations and Java similar to publishing adapters. Hence, they may have the same problems with JMS Queues, Java CLASSPATH, or JNI DLL configuration entries as the RIB publishing adapters. However, TAFRs do not typically involve database operations. On the other hand, TAFR adapters may have their own configurations specified in *property files* that detail the transformations or routing that must occur.

Fatal TAFR failures will cause a message backlog in the source JMS queue. TAFRs with incorrect routing logic will route messages to incorrect destinations.

SeeBeyond deployed Subscribing adapter failures

Subscriber adapters have the same Java, JNI DLL, and Connection Point potential problems as publishing adapters. When these problems occur, messages are not delivered to the adapter and the source message queue will become backlogged.

However, subscribing adapters may also run into problems due to the field content of the messages. For example, there may be a mismatch with a value or ID found in the message. When this occurs, the following takes place:

1 The subscribing adapter keeps track that the message failed internally and returns a failure to the e*Gate system.

2 A distribute rollback is performed. All database work is rolled back and the message remains on the source JMS queue.

3 The message is re-processed. Because the adapter has flagged the message has failed, it inserts the message into the Error Hospital.

4 A distributed commit is performed. The message is removed from the source queue and is committed to the Error Hospital.

5 Periodically, a second collaboration associated with the Error Hospital awakens and pulls the data from the Error Hospital. This collaboration then inserts the message back into the original source queue.

6 Steps 1-5 are repeated until the message is successfully processed or until maximum retry count is reached.

Note that both a GUI as well as a command line interface are provided to administer the Error Hospital. Error Hospital operations are detailed later in this manual.

Deployment architecture considerations

So far, the components have been described in generic terms. This is because every installation may have its own unique configurations and needs. However, there are some configuration patterns or philosophies that Retek suggests for successful RIB operations.

Retek schema integrity on the SeeBeyond Platform

Retek suggests that the messaging schema supplied with the Retek Integration Bus be modified as little as possible when deployed to a production environment. Doing so will ease the pain of installing RIB updates. Each future RIB release is expected to contain additional application integration points and Message Families. Segregating the Retek messaging schema from other non-Retek components will enable updates to be installed quicker and with fewer side effects.

Disk space analysis

Before the RIB is deployed to production, an analysis of the expected message traffic must be made. The Retek 10.3 Integration Guide lists all of the messages as implemented within the RIB and the conditions in which they are published. System designers use this guide to estimate expected message size and volume. From a business operations viewpoint, one should also determine the amount of time a specific subscriber is allowed to be unavailable before serious business consequences occur. This should include the maximum amount of time before a subscriber is failed-over to another system.

The purpose behind this analysis is to determine the amount of disk space needed to support continued operations if a subscriber becomes unavailable. The standard RIB configuration will maintain a copy of each message on a queue's persistent storage until all subscribers have processed the message. If the disk –subsystem or queue's configuration cannot store messages, then each publisher will need to be shut down.

This analysis should also be continued to the publisher. Specifically, Retek suggests performing these calculations on the Message Family Manager staging table size and the likelihood of the SeeBeyond EAI system becoming unavailable for a specific amount of time. In this scenario (which may be a continuation of a subscriber problem) the publishing e*Way may not be able to publish messages. As such, all messages become backed up in the MFM staging table. If this table runs out of space, then all application triggers that write to the table will fail and the application should be shutdown.

Intelligent queue managers

The SeeBeyond e*Gate EAI platform allows one to use a number of different Intelligent Queue Managers for storing published messages. The Retek Integration Bus is designed to use JMS queues because this component requires no external database and implements the XA interface protocol. The XA protocol enables the “exactly once” message processing.

The purpose of an IQ Manager is to manage Intelligent Queues. In most cases, these queues are explicitly defined. In the case of the JMS IQ Managers used with the RIB, explicit queue definition is not needed. The JMS IQ Manager also provides a JMS Service to the Connection Point interface. Each event type published using the JMS Service will use the Event Type name as the JMS “topic”. The configuration of the JMS service sets other parameters needed to access the message.

Note: Not only Java Collaboration Rules can be used with JMS Connection Points. Monk Collaboration Rules can publish/subscribe to messages on a JMS queue, but must also explicitly define a JMS Intelligent Queue on the JMS IQ Manager used.

Performance motivated parallel processing

A common method to gain throughput in distributed EAI systems is to duplicate processing modules across multiple systems or, if the system spends a significant percentage of time waiting for disk I/O, to duplicate modules within the system. These components then execute in parallel, reducing the elapsed time for processing multiple messages.

In the Retek 10.3 release, parallel processing considerations have been subordinated to message sequencing guarantees. In other words, the design of the system guarantees message processing is in the correct sequence as opposed to maximizing throughput.

Additional throughput gains can be made if the system is deployed with parallel processing nodes. However, simply duplicating these nodes introduce the possibility that some data will be processed out-of-order. If this occurs, then the final state of the subscribing system will be incorrect and contain invalid data.

Thus, additional design and implementation work is needed to support parallel processing deployments of the RIB in the 10.3 release. This work must center on creating well-defined logical channels of information, each channel responsible for a well-defined set of business entities. An example of such a logical channel would be one responsible for all of the "even numbered" purchase orders. This is similar to the Retek “Batch Thread” model. Briefly, the following changes would need to be made:

1 The current message flow (Publishing adapter and all TAFRs and subscribing adapters) would need to be duplicated once per each logical channel.

2 For each publisher, the MFM Oracle database package would need to be modified such that the “GETNXT()” procedure only returns messages concerning a subset of all available business entities. If two publishers were used, then one would return only even IDs and one, only odd IDs.

3 Additional configuration changes would be needed to insure that different Error Hospitals are associated with each new subscriber.

4 Each logical channel *should* have an associated Connection Point that uses a distinct JMS Service provider. This involves creating a JMS IQ Manager for each logical channel and a JMS Connection Point that uses this JMS IQ Manager. This JMS Connection Point would then be the source or destination for all messages on the channel. Otherwise, the messages published for one channel would become intermingled with those from other channels when the JMS provider saved them to stable storage.

An alternative to multiple JMS IQ Managers is to rename all of the event types used within the logical channel to be channel specific.

Chapter 3 – Configuration files

The various RIB platforms leverage some platform specific configuration mechanisms. However, most RIB specific parameters are specified in a file known as the RIB Properties file.

The Retek Binding sub-system is used on the ISO and J2EE environments. It uses its own set of configuration files for determining the code to execute when publishing or subscribing to a specific message family.

RIB Properties File

The RIB Properties File has the name `rib.properties`. Its location on the system is dependent on the deployment of the RIB and the running system's CLASSPATH specification. See each platform's configuration chapter for more details.

This section details the contents of this file.

RIB Logging and Timings File

This section details the file names and levels of logging (on/off or normal/verbose) for RIB File logging and Timings logging. A Timings log file contains a series of timestamp lines the mark the date and time a processing point has reached. Multiple threads may write the same Timings log file. A post processor is needed to determine statistics about the running system.

log.default.file_path – Location for the rib log files to be places on the server.

log.default.verbose – Default logging to use if none specified for an adapter

log.<adapterName>.timings – If this property is set to 'Y', then a timings log is created and logged during the execution of the adapter specified.

log.<adapterName>.timings_logFile - When the timings is set to 'Y', this specifies the file time stamp entries are written to.

Log4j support in the RIB Properties File

In addition to the functionality and properties described in the previous section; the RIB, starting with the 10.3.4 release, takes advantage of the Apache Software Foundation's log4j logging system.

Introduction to log4j

Log4j is a logging service allowing users to specify at runtime the granularity of the information displayed, the format, and the destination of logged data. It is fully customizable through properties in a configuration file. This information can be displayed to the standard output, a file or a remote service (e.g.: JMS, E-mail, TCP). Many properties exist for configuring the location of the log files and the content that each one should store (level or relevance of the information, as well as where in the program the information has been originated). It can also control other attributes of the file containing the log entries, such as

- the maximum size of the log files
- whether to roll over to a new file based on file size or time criteria
- number of backups of previous files

To learn more about log4j, visit the Apache Software Foundation's URL (http://logging.apache.org/log4j/docs/documentation.html) and click on the "short manual," "javadoc documentation," and "FAQ" links.

New log properties in the rib.properties file

In 10.3.4, the RIB comes configured to use log4j. However, this configuration duplicates the behavior seen in previous RIB releases. Any translation between rib and timings properties and log4j properties occurs in the RIB infrastructure code. However, in order to maximize the benefits of log4j a user must replace the pre-existing log properties with properties supported by the log4j service. This is an automatic process that just requires the execution of a tool provided with the RIB. The following command will convert the existing rib.properties file to support log4j properties:

```
java -DEHOME=$EHOME –cp $EHOME/client/classes/retek-rib-support.jar
com.retek.rib.log.converter.RibLogToLog4jPropertiesConverter
```

This command must be run locally in the machine hosting the eGate registry. A backup copy of the original rib.properties file will be preserved as rib.properties.bak. One of the first things a user will notice is an increase in the size and content of the new rib.properties file. More properties per adapter are needed by log4j in order to provide the user with flexibility of configuration. Lets look at the ewStoresFromRMS e*Way log properties as an example.

For RIB Logging, the single `log.ewStoresFromRMS.verbose` property has been substituted with the following set:

```
log4j.logger.rib.ewStoresFromRMS=INFO, ewStoresFromRMS_appender
```

This property defines the component responsible for logging information from the ewStoresFromRMS e*Way. The first value after the equal sign, indicates the degree of information to be logged. In the example, it indicates that it should log messages down to the INFO level. The levels available and their relevance from lowest to highest are ALL < DEBUG < INFO < WARN < ERROR < FATAL < OFF. A level of INFO will log messages designated as info, warning, error or fatal, but will leave out any debug messages. ALL and OFF, although not really levels, are used as switches to turn completely on or off the logging of events. The second value, known as the appender, indicates the destination of the messages to be logged. Multiple appenders can be specified by providing a comma-separated list. The list of appenders can represent any combination of standard output/error, files or remote services.

```
log4j.additivity.rib.ewStoresFromRMS=false
```

The additivity flag indicates whether the logger should log messages to just the appenders set in the `log4j.logger.rib.ewStoresFromRMS` property or to other appenders set above in its hierarchy. Please refer to the log4j documentation from the Apache Software Foundation for a discussion of logger hierarchy and the additivity flag.

Appenders are configured using an additional set of properties. Appenders control the attributes of the destination log file.

```
log4j.appender.ewStoresFromRMS_appender=
          org.apache.log4j.RollingFileAppender

log4j.appender.ewStoresFromRMS_appender.File=
          ${log.default.file_path}/rib_ewStoresFromRMS.log
```

```
log4j.appender.ewStoresFromRMS_appender.MaxFileSize=1024KB
log4j.appender.ewStoresFromRMS_appender.MaxBackupIndex=10
```

In the properties listed above, the ewStoresFromRMS e*Way will log messages to the rib_ewStoresFromRMS.log file up to a maximum of 1Mbyte (the size is actually slightly larger, depending on the size of the last message logged required to reach the specified size). After this value has been reached, a new rib_ewStoresFromRMS.log file will be created up to a maximum of 10 backup copies. The old log files will be renamed rib_ewStoresFromRMS.log.1, rib_ewStoresFromRMS.log.2, rib_ewStoresFromRMS.log.3 and so on. The most recent backup will have the lowest index (1) while the oldest one will have the highest (10). After the maximum number of backups is reached, older files will be deleted and only the ten most recent backups will be kept.

```
log4j.appender.ewStoresFromRMS_appender.layout=
          org.apache.log4j.PatternLayout
log4j.appender.ewStoresFromRMS_appender.layout.ConversionPattern=
          %d{yyyy.MM.dd HH:mm:ss.SSS}|%m%n
```

These appender properties indicate the format to apply to each message right before inserting it into the appender. One of the most powerful functionality of log4j is the ability to control at runtime the way the information is presented to the user; therefore, giving him/her the ability to ultimately decide how he/she wants the messages displayed.

In the same manner, the Timings properties:

```
log.ewStoresFromRMS.timings=N
log.ewStoresFromRMS.timings_logfile=
      timings_ewStoresFromRMS.log
```

has been substituted with the log4j properties:

```
log4j.logger.timings.rib.ewStoresFromRMS=
            OFF, timings_ewStoresFromRMS_appender
log4j.additivity.timings.rib.ewStoresFromRMS=false
log4j.appender.timings_ewStoresFromRMS_appender=
        org.apache.log4j.RollingFileAppender
log4j.appender.timings_ewStoresFromRMS_appender.File=
    ${log.default.file_path}/timings_ewStoresFromRMS.log
log4j.appender.timings_ewStoresFromRMS_appender.MaxFileSize=2048KB
log4j.appender.timings_ewStoresFromRMS_appender.MaxBackupIndex=10
log4j.appender.timings_ewStoresFromRMS_appender.layout=org.apache.lo
g4j.PatternLayout
log4j.appender.timings_ewStoresFromRMS_appender.layout.ConversionPat
tern=%d{yyyy.MM.dd HH:mm:ss.SSS}|%m%n
```

Two new properties have also been added:

```
riblog.suppress.repeated.message=true
riblog.suppress.repeated.message.counter=100
```

Together they help to control the unnecessary logging of repeated information that is caused in periods of inactivity during the lifetime of the e*Way. If a message has been logged already, and the `riblog.suppress.repeated.message` is set to true, the same message will not be logged again until the 101st time (or the value of `riblog.suppress.repeated.message.counter + 1`) at which point the message is logged only once an the same process repeated.

Enabling log4j for e*Ways

New in 10.3.4, all e*Ways are required to have the log4j.jar added to their classpath. Failure to comply with this requirement will cause a message similar to the next one and the immediate shutdown of the e*Way:

```
java.lang.NoClassDefFoundError: org/apache/log4j/Category
```

Please refer to the 10.3.4 Release Notes for instructions on how to update the e*Ways to include the log4j.jar entry in their classpath.

RIB Message bundling entries

<eway name>.<collaboration name>.pubMessageCount – This attribute is used to determine the number of times the publishing thread will attempt to call the GETNXT() stored procedure within a single transaction. It also specifies the maximum number or RIB Message Nodes that can be included in a single <RibMessages> tag. This is a new property in the 10.3 release.

This property is optional. If not specified, it defaults to 1. This is a performance tuning property that can reduce the amount of time spent between collaboration calls and also reduce the frequency of committing data to JMS and Oracle.

Multi-threading entries

This section details those entries used to support multi-threading within a message family. Multi-threading allows simultaneous processing among multiple threads of control for messages within the same message family. If performed correctly, this allows for large throughput gains while still maintaining the RIB's sequencing and exactly once guaranteed processing.

mfm.<family name>.total_threads -- defines the total threading level to be used but not exceeded by this message family.

mfm.<family name>.<collaboration name>.thread_num – defines the specific thread number that the specific collaboration is to use upon execution.

Note that upon start up of some publishing e*Ways there is a synchronization check with the database on the total_threads in rib.properties, and if the data is not the same the e*Way is shutdown without processing any data, as the publishers algorithm for deciding what data to publish to each publisher may be dependent on the threading value configured.

Error Hospital entries

This section details the entries used for retrying messages from the Error Hospital.

hospital.attempt.max – This is the maximum number of attempts to try to push this record through the RIB automatically, once this retry count is exceeded the message remains the Error Hospital DB but is no longer retried automatically.

hospital.attempt.delay – value (in seconds) used to calculate the next attempt time

hospital.attempt.delayIncrement – value (in seconds) used to calculate the next attempt time.

The next attempt time is calculated as:

```
hospitalAttemptDelay + (hospitalAttemptDelyIncrement * attempt
count)
```

This is done so that the delay between each attempt is longer than the previous delay.

Global entries

dtd_url.default – Specifies the DTD File location. RIB Payloads include a DOCTYPE specification.

default.MessageSelectorCheck –When this value is set to 'true', all e*Ways that subscribe to JMS topics will verify that their message selector is set up properly on their durable subscriber within the SeeBeyond JMS server.

default. SubscriberCheck –When this value is set to 'true', all e*Ways that publish to JMS will verify that a subscriber exists for the specified topic within the SeeBeyond JMS server.

Implementation classes used

In order to promote pluggable, platform specific implementations, the RIB allows the specification of platform-specific classes for a variety of functions. These functions include the actual creation of a RibMessages XML message and the interface to an alert mechanism. The following entries are used to specify what Java classes should be used for these functions:

alertPublisherImpl -- Interface to the Alerting mechanism

```
Values: com.retek.rib.sbyn.alert.EgateAlertPublisher (SeeBeyond)
```

ribMessageImpl – Class used to create a ribMessage node within a RibMessages container.

```
Values: com.retek.rib.sbyn.RibMessageWrapper  (SeeBeyond)
```

ribMessagesImpl – Class used to create a RibMessages container.

```
Values: com.retek.rib.sbyn.RibMessagesWrapper  (SeeBeyond)
```

routingInfoImpl – Class used to create the Routing Information Section within a ribMessage node.

```
Values: com.retek.rib.sbyn.RoutingInfoWrapper  (SeeBeyond)
```

failureImpl – Class used to create, store and copy message failure information

```
Values: com.retek.rib.sbyn.FailureWrapper (SeeBeyond)
```

subListForTopicImpl – Implementation class for listing subscribers for a topic utilizing the JMS vendor's utility command.

```
Values: com.retek.rib.sbyn.cmd.SubListForTopic (SeeBeyond)
```

topicStatImpl – Implementation class for listing topic statistics utilizing the JMS vendor's utility command.

```
Values: com.retek.rib.sbyn.cmd.TopicStat (SeeBeyond)
```

deleteSubImpl – Implementation class for removing a subscriber from a topic utilizing the JMS vendor's utility command.

```
Values: com.retek.rib.sbyn.cmd.DeleteSub (SeeBeyond)
```

SeeBeyond platform specific entries

This section details the SeeBeyond platform specific entries

eway.<e*Way Name>.no_event_sleep_millis – This entry specifies how much time to sleep when no information is available to be published for a specific e*Way. The actual e*Way name must replace the string <e*Way Name>

eway.default.no_event_sleep_millis – This entry specifies how much time to sleep when no information is available to be published and there is e*Way specific no_event_sleep_millis entry.

Application specific entries

RDM specific entries

facility_type.default – Specifies the default facility type to be used by RDM publishing e*Ways for calls to RDM.

facility_id.<facility_type>.<location id> - This property is used by the routing TAFRs to determine which RDM topic to route a message to based on the facility type and location id used.

<eway name>.<collaboration name>.dc_dest_id – Used by RDM publishers as input parameters to the Oracle DB requests. Should be set to the appropriate DC Destination ID for the data that is desired from the RDM instance being connected to.

multichannel_ind – this field has been deprecated (a.k.a. no longer used).

ISO platform specific entries

There are no entries for ISO that are any different from the normal SeeBeyond entries. Only a small subset of the entries for SeeBeyond Rib components, however, are required in the rib.properties file for the Rib ISO components. These are the entries for the error hospital, as the Rib ISO components still makes use of the error hospital, and entries for the implementation classes used.

FlowTrak specific entries

prop.strm.fname - Location of the FlowTrak properties file.

Retek Binding configuration files

Properties files

payload.properties – The purpose of this file is to map a RIB message family and RIB message type key to the fully qualified class name of a Java payload object. A Java payload object is a JavaBean-like object, with a number of attributes that can store data from an XML document. Each of these payload objects is inherited from a common Retek Binding Payload super class, the com.retek.binding.rib.payload.Payload Object. This file is used in both publishing and subscribing APIs, as both use a Payload objects.

Properties in this file should be entered as follows:

(message family).(message type) = fully-qualified Java class name

```
ASNOUT.ASNOUTCRE=com.retek.binding.rib.payload.ASNOutDesc)
```

binding.properties –The purpose of this file is to map a RIB message family and RIB message type key to a Castor-generated mapping file. The mapping file is specific to the DTD describing the format of the message payload. This file is by default empty, as the RIB does not use Castor Mapping files for XML Binding. However, the ability to use the Castor Mapping file is available by entering in a new property for a certain message family and type in this file.

The properties in this file are entered in the same way as the payload.properties file:

(message family).(message type) = relative path to Castor Mapping file

```
ASNOUT.ASNOUTCRE=com/retek/binding/rib/payload/ASNOutDescMap.XML
```

For more information on the Castor mapping files, see the “XML Files” section, below.

XML files

<PayloadObjectName>Map.xml –These Castor-generated Mapping files map the element names in an XML schema document, to the attribute names and methods in a Java Payload object. By default, the RIB does not use Castor Mapping files for Binding, as it uses the Castor-generated Descriptor files to create the mapping between the XML Schema and Java Payload Object. A Castor Mapping file can be used in the RIB by defining the relationship in the binding.properties file. If a value is defined for a certain message family and type, that value will be used as the path to the Mapping file and will be used for XML Binding with Castor.

These Mapping files need to reside in the retek-payload.jar, along with the Java Payload objects. The path to these files in this jar needs to match the path entered in the binding.properties file. For example, if the properties file defines the mapping as “com/retek/rib/binding/payload.ASNOutDescMap.xml”, this would need to be the path to this file inside the retek-payload.jar.

Chapter 4 – SeeBeyond Platform

RIB startup and shutdown

This section details how to start up and shut down the RIB.

Available Scripts

Bringing up and down the RIB can be done using a series of scripts supplied with the RIB. These start and stop scripts are:

start_egate	starts the SeeBeyond registry
start_cb	starts the SeeBeyond control broker for the RIB Schema
start_rib	starts RIB JMS IQ Manager(s) and e*Ways in a known sequence.
stop_rib	stops RIB JMS IQ Manager(s) and e*Ways in a known sequence.
stop_cb	stops the SeeBeyond control broker for the RIB Schema
stop_egate	stops the SeeBeyond registry

In general, one should start up the components in the following manner:

First the registry, then the control broker, then the JMS IQ Manager, then the subscribing e*Ways, then the TAFR e*Ways, then the publishers.

In a standard installation, the start_egate script will reference a file named *egate.txt.* This file contains all of the standard e*Ways and JMS IQ managers that come with the RIB schema. If invoked with the "-f <filename>" switch, this script will use the supplied control file for determining which e*Ways and JMS IQ Managers to bring up or down. A complete listing of options for the start_rib script is found below. Similar execution options are available for the stop_rib script.

```
start_rib [-r] [-s schema_name] [-f eway_file] [-u user_name] [-p
user_password] [-e eway_name] [ALL] [JMS] [SUB] [TAFR] [PUB] [HOSP]

Where

    -r                specifies to create/update the
"failed_eways.txt" file with the names of the elements not booted

                      This file may be used with the -f switch on a
later execution.

    -s schema_name    specifies the name of the schema to start --
default is RIB103

    -f eway_file      specifies the file containing eway
description, default is $EHOME/RIB/eways/eways-out/Egate.txt

    -u user_name      specifies the user name to use -- default is
Administrator

    -p user_password  specifies the password to use -- default is
STC

    -e eway_name      specifies only a single eway or other element
to start

    ALL               specifies bringing up all elements listed in
the eway file. Equivalent to JMS SUB TAFR PUB HOSP

    JMS               specifies bringing up all JMS elements listed
in the eway file

    SUB               specifies bringing up all SUB (subscriber)
elements listed in the eway file

    TAFR              specifies bringing up all TAFR  elements
listed in the eway file

    PUB               specifies bringing up all PUB (publisher)
elements listed in the eway file

    HOSP              specifies bringing up all HOSP (hospital)
elements listed in the eway file
```

The format of the eway_file (typically Egate.txt) is:

```
            <name>   <type>   <seq>
```

Where `<name>` is the name of the element/JMS/e*way, `<type>` is one of JMS, SUB, TAFR, HOSP, PUB, `<seq>` is a number detailing the order of operations within a type. Starting is performed in ascending order. Stopping is performed in descending order. Comment lines must begin with two forward slashes, "//"

Sequencing considerations – Detailed Information

In the RIB architecture, the first step a Retek application performs in publishing a message is the execution of a table specific trigger. These triggers are installed in a disabled state with each application. See the Retek Integration Bus Installation Guide or the product specific installation guide for information on the triggers and how to enable them.

The SeeBeyond EAI components can be configured to come up manually or automatically. If configured to be brought up automatically, then only the registry and control brokers need to have an external method for starting. On Unix systems, this method is typically found in a startup script executed when during the system boot sequence. The components run as daemons.

Note: Sample scripts to start the registry and control broker can be found in the $EHOME directory. This is the directory where e*Gate was installed and was configured as part of the RIB installation process. "start_egate" and "start_cb" are the two scripts to refer to.

A generalized list of steps needed to start an e*Gate system is found below. Complete documentation on SeeBeyond e*Gate operations is found in the SeeBeyond e*Gate Integrator System Administration and Operations Guide. Please refer to this manual for further information on the referenced commands.

1 Open all external resources that the components are dependent on, such as an application's database.

2 Open the SeeBeyond e*Gate Registry.

- If the RIB Installation Instructions were followed, run the "start_egate" script from the $EHOME directory and skip to step 6.

  or

- On Unix systems, this is done via the `stcregd` command.

3 Before the `stcregd` command may be executed, initialize the user's environment correctly. This is typically performed by "sourcing" the file $EHOME/server/egatereg.sh.

Note: If the RIB Installation Instructions were followed, this step is done by the "start_egate" script.

For example, for Korn or Bourne Unix shells:

```
> . $EHOME/server/egatereg.sh
```

4 The parameters needed for the `stcregd` command specify the registry's name and TCP port numbers. It is suggested that only one registry be configured for a host, as this simplifies the configuration of the startup script for the registry and control brokers. However, site-specific issues may motivate an EAI administrator to configure multiple registries on the same computer.

Note: Examples of such issues include using a test system as a "hot standby" for a production system, or providing extra redundancy for the registry on the local system.

5 The following `stcregd` command displays a registry named “egate_main” using the default TCP ports for the initial connect port and the connections made between the registry and control brokers. It also executes *without* Access Control Lists used for authorization purposes:

```
> stcregd –ln egate_main
```

Switches for this command include:

- -pr Port number for Registry Clients
- -pc Port number for Control Brokers
- -ln Registry logical name
- -mc Maximum number of connections
- -bd Base directory
- -ss Run as a service
- -h Display help screen

SeeBeyond suggests that the name of a registry matches the name of its host computer.

6 Open the control brokers for all participating hosts.

- If the RIB Installation Instructions were followed, run the “start_cb” script from the $EHOME directory and skip to step 11.

or

- On Unix systems, this is done via the `stccb` command.
- On Microsoft Windows platforms, the registry is typically installed as a service.
- The `stccb` command is also available as a DOS command.

7 Before the `stccb` command may be executed, the user’s environment must be initialized correctly. This is typically performed by “sourcing” the file <EHOME>/server/egateclient.sh.

Note: if the RIB Installation Instructions were followed, the “start_cb” script does this step.

For example, for Korn or Bourne Unix shells:

```
>  . $EHOME/server/egateclient.sh
```

8 An **stccb** daemon must be running for each participating host *on* that participating host.

9 The parameters needed for the **stccb** command specify the control broker’s name and TCP/IP address of available primary and secondary registries.

10 The following **stccb** command brings up a control broker with the following attributes:

- Named “cb_main”
- Contained the schema “RIB102”
- Uses the registry found on the host “egate_main” with the default TCP port numbers
- Runs under the SeeBeyond e*Gate defined “Administrator” user-id
- Authenticates itself to the registry using the password “STC”

Note: This is the commonly used “Default” password for SeeBeyond e*Gate installations. Any installation wishing to provide even a modicum of security will change this password. Furthermore, the password may be encrypted and stored in a file via the `stcutil` command, so that it is not visible to casual observers. See the SeeBeyond e*Gate Integrator System Administration and Operations Guide for more details.

```
stccb –ln cb_main –rh egate_main –rs RIB102
–un Administrator –up STC
```

- Executes *without* Access Control Lists used for authorization purposes.11 At this point, you can display the e*Gate Monitor application to start any components not configured to be brought up automatically. This application requires a Microsoft Windows platform for execution.

12 Using the e*Gate Monitor, display all of the JMS Queue Managers needed.

13 Using the e*Gate Monitor, display all of the e*Ways and / or schema bridges. Adapters that subscribe to messages and interface directly to an application should be brought up before those that publish messages.

RIB message publishing adapters

Adapters that publish messages directly from Retek applications have names in the following format: `ewMSGFAMILYFromAPPNAME`, where `MSGFAMILY` is the name of the message family published and `APPNAME` is the name of the publishing application, such as RCOM, RMS, RDM or RDC.

For a listing of all the available publishing adapters, refer to the RIB 10.3 Integration Guide.

RIB message subscribing adapters

Adapters that subscribe to RIB messages and update Retek applications have names in the following format: `ewMSGFAMILYToAPPNAME`, where `MSGFAMILY` is the name of the message family published and `APPNAME` is the name of the publishing application, such as RCOM, RMS, RDM or RDC.

For a listing of all the available subscribing adapters, refer to the RIB 10.3 Integration Guide.

TAFR adapters

TAFR adapters process messages in support of subscriber specific needs. As such, they are both subscribers and publishers. TAFR Adapters have names in the following format: `ewMSGFAMILYToMSGFAMILYFromRIB`, where `MSGFAMILY` is the name of the message family the TAFR works on as input, `ToMSGFAMILY` is the name of the message family the TAFR publishes and `APPNAME` is the name of the final subscribing application.

For a listing of all the available TAFR adapters, refer to the RIB 10.3 Integration Guide.RIB error hospital

The RIB error hospital is a subsystem used to retry messages the subscriber has failed to process successfully. After a failure, the message is inserted into the hospital database associated with the subscriber. This message is then republished a configurable number of times by a “retry” collaboration. The “retry” collaboration is also found within the subscriber adapter and is only responsible for re-publishing the message.

The Error Hospital may also contain messages that are dependent on a “failed” message. The dependency is based solely on a common business entity that the two messages reference. For example, if a “Create New PO” message fails (and is added to the hospital), then a subsequent “Add PO Line Item” will also be added to the hospital if it references the same PO. The “retry” collaboration will resubmit both messages in the correct order.

The RIB message error hospital requires that the “Retry” collaboration is included within a subscribing e*Way and uses a valid connection point as the source of its retry events.

The database tables comprising the Error Hospital storage may be found within the same database as the stored procedures called by the subscribing adapter or in a separate database. If the error hospital tables become inaccessible, then any failing message will cause the total stoppage of all messages by the subscriber. This consideration should be taken into account when determining the location of an Error Hospital for a subscriber.

Preventative maintenance tasks

This chapter lists some common tasks that a system administrator may want to script and perform on a regular basis.

Log files

The SeeBeyond e*Gate EAI system can log volumes of data to log and journal files. Furthermore, because the RIB uses two phase commit, the SeeBeyond system, acting as the transaction manager, must log commit information within “transaction log” files in order for distributed transaction recovery purposes.

E*Gate’s error, trace, and debug log files

The same file is used by SeeBeyond e*Gate adapters for logging error messages, trace messages, and debugging messages. The adapter’s configuration determines what is to be logged and the level of logging. If logging is turned on, then the free disk space should be closely monitored, as these files can rapidly increase in size and grow to enormous sizes, even if the e*Way has only processed a relatively few messages.

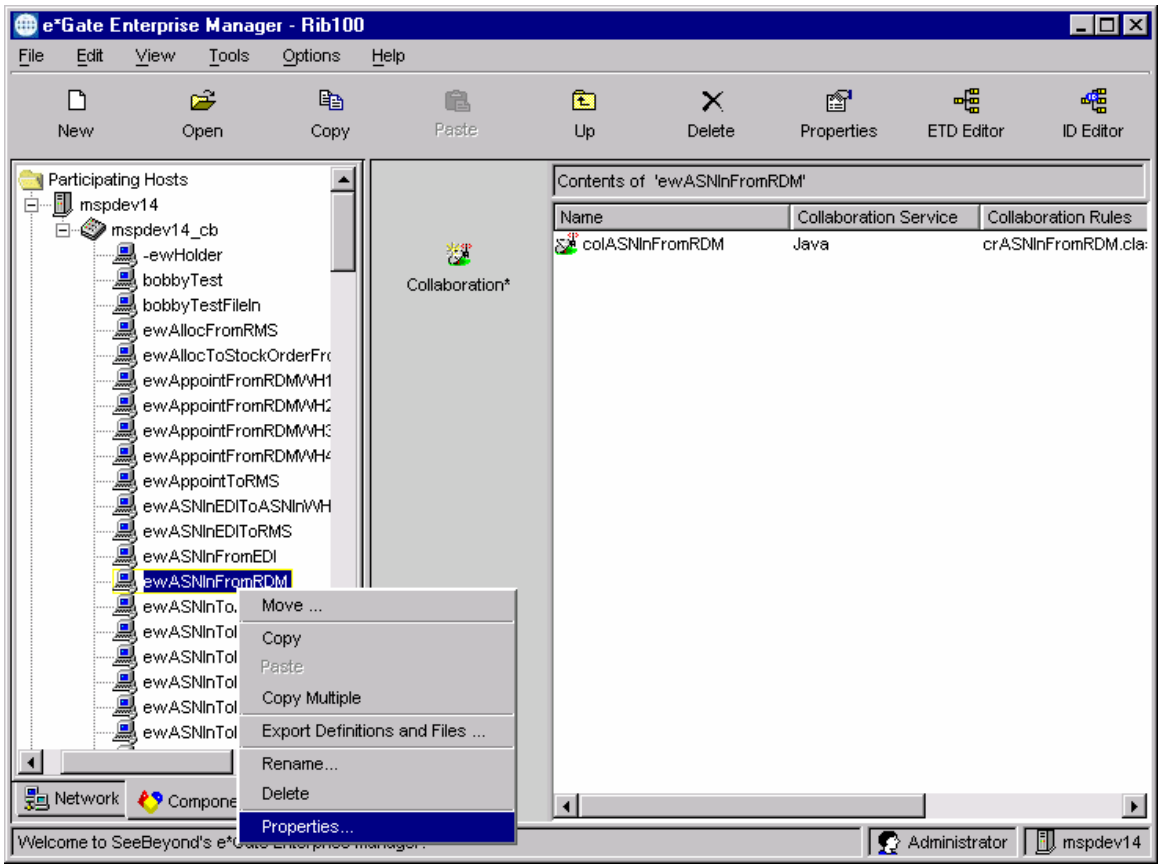
The location of the log files is the directory <EHOME>/client/logs, where <EHOME> is the installation directory for the SeeBeyond e*Gate EAI system. Each component has its own log file named <component>.log, where <component> is the name of the e*Way, control broker, or IQ Manager.

Additionally, there may also be files containing application "standard error" output. These files are named <component>.stderr .

Sometimes it is helpful to have component log information to determine a problem's source or otherwise monitor its activities. The e*Gate Enterprise Manager application is used to modify level and type of logging for an e*Way. Further information may be found in the SeeBeyond e*Gate Integrator User's Guide.

To turn on, and/or modify, SeeBeyond's e*Gate adaptor logging:

1 The first step is to select the RIB adapter component from the main e*Gate Enterprise Manager window:



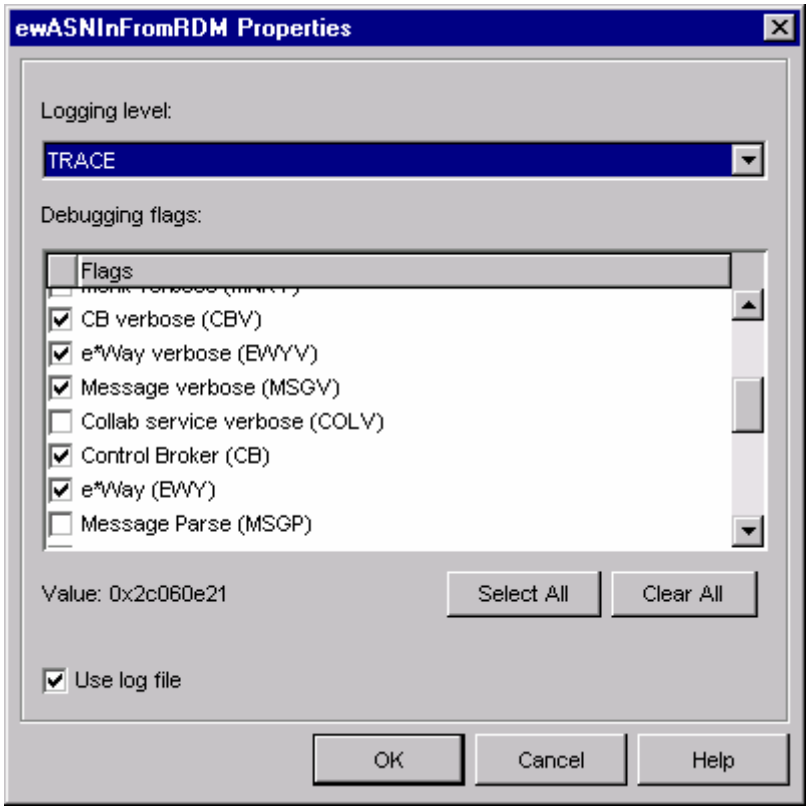
Selecting an e*Way from the e*Gate Enterprise Manager

2 Right click on the e*Way.

3 Select Properties. The Properties window is displayed:

4 Click on the Advanced tab.

5 Click **Log**.



e*Way Logging window

There are two dimensions to e*Way logging: the areas of information that the log entries will log about, and the amount or level of logging. There is only one level of logging for all areas.

Over 25 different areas are available for logging.

To log RIB Adapter-created messages:

6 Select the e*Way (EWY) check box to enable logging.

7 In the Logging File field, select TRACE.

8 Select the Use Log file check box.

Be careful whenever logging is enabled, as log files are not limited in size and can grow to be quite large. In normal production, you should set the logging level to be at a very low level: either “FATAL”, “ERROR”, or “NONE”.

RIB logger

The RIB has its own logging capabilities. The RIB support Java classes contain logging logic which write to RIB log files. The rib log filenames are in the format “rib_<ewName>.log” and are written to a user specified directory. Additionally, the RIB logger has the ability to generate a timings log that can be used to measure performance.

rib.properties log entries

The following are the entries in the rib.properties file which pertain to the RIB logger:

Path where RIB and Timings log files will be written. It must end with a directory separator / or \.

log.default.file_path=/files0/egate/RIBLOGS/

Log e*Way times? [Y or N]

log.<ewName>.timings=N

File to write timings log entries to. Only specify the file name, as it will be pre-pended with the log.default.file_path property.

log.<ewName>.timings_logfile=timings_<ewName>.log

Default logging level verbose? [Y or N]

log.default.verbose=N

e*Way specific logging level verbose? [Y or N]

log.<ewName>.verbose=N

XA transaction log files

Whenever a two phase commit operation commences, the transaction manager (TM) must log the decision to commit the transaction to stable storage. This is to insure the transaction will commit if a failure occurs during the second phase. These "log_commit" records are read whenever a TM is started so all-active transactions are completed.

The SeeBeyond e*Way implements a transaction manager. The transaction log record for collaboration is found in its own file. The path name of the file is:

```
<EHOME>/client/XALogs/<e*WayName>/<collabName>
```

Where `<EHOME>` is the installation directory for the e*Gate product, `<e*WayName>` is the name of the e*Way the collaboration runs in, and `<collabName>` is the name of the collaboration.

Do not delete these transaction log files. If these files are deleted, then the adapter associated with the log file(s) may have problems re-processing messages found in the error hospital or even completing initialization successfully.

If a database or other resource manager has a transaction in a prepared state and the associated transaction log file is deleted, then the database or resource manager also must have its knowledge of the transaction removed.

For Oracle databases, transactions that are in the prepared state can be found in the DBA_2PC_PENDING views. One can then use an external database session, such as one with the SQLPLUS command, to force a rollback or commit operation on these transactions.

MFM staging tables

Part of the RIB's architecture is that data is staged from applications using database tables. The RIB adapters use a well-defined interface to retrieve this information when the publishing it to the RIB.

The code that wrappers access to these staging tables is known generally as the Message Family Managers (MFMs). The MFM implements the interfaces for extracting the data as procedures found within an Oracle database package. For more information on MFMs in general, see the Retek Integration Bus Technical Architecture Guide. For information about a specific MFM, see the Retek 10.3 Integration Guide.

Some MFMs require that data in the staging table from multiple application transactions be coalesced into a single message. In these cases, the MFM waits until a specific record is inserted into the staging table before the message is published. For example, new Purchase Orders may not be published until they have been placed into an "approved" state.

A system administrator may monitor the MFM staging tables to verify that the RIB's performance is adequate to handle the messaging traffic. If a system has the adequate resources, then the number of rows within the staging table should remain relatively constant.

Error Hospital

Subscribing Error Hospital

The RIB error hospital is a subsystem used to retry messages the subscriber has failed to process successfully. After a failure, the message is inserted into the hospital database associated with the subscriber. This message is then republished a configurable number of times by a "retry" collaboration. The "retry" collaboration is also found within the subscriber adapter and is only responsible for re-publishing the message.

The Error Hospital may also contain messages that are dependent on a "failed" message. The dependency is based solely on a common business entity that the two messages reference. For example, if a "Create New PO" message fails (and is added to the hospital), then a subsequent "Add PO Line Item" will also be added to the hospital if it references the same PO. The "retry" collaboration will resubmit both messages in the correct order.

The RIB message error hospital requires that the "Retry" collaboration is included within a subscribing e*Way and uses a valid connection point as the source of its retry events.

The database tables comprising the Error Hospital storage may be found within the same database as the stored procedures called by the subscribing adapter or in a separate database. If the error hospital tables become inaccessible, then any failing message will cause the total stoppage of all messages by the subscriber. This consideration should be taken into account when determining the location of an Error Hospital for a subscriber.

Publishing Error Hospital

In the 10.3 release, a new publishing paradigm was introduced for enhanced performance. This design uses referenced application data instead of copied data. The work used to extract the data was also moved from the application triggers to the GETNXT() stored procedure. However, this design allows for the possibility that the data may be locked or otherwise unavailable when GETNXT() is called. When this occurs, the application may request the e*Way to insert a row into the Publisher Error Hospital, where another attempt to publish the data may be made later. This allows the subsequent call to the oracle publisher to process the next message and not get stuck trying to retry a flawed record in its staging table over and over.

A publishing e*Way will check the error hospital for previous message data that is currently in the Error Hospital for the Business Object (e.g. PO) the current message is publishing. If such a dependency exists, the dependant message is put into the hospital as well.

The Publisher Error hospital is facility uses the same database tables as the Subscriber Error Hospital. However, an additional Publishing Retry e*Way has been created for each application that request an insert into the Error Hospital.

The publishing retry e*Way processes data differently than a subscribing retry e*Way.

A major difference is that the publishing retry e*Way retries directly to the database, although it calls the same package, it calls a different procedure, PUB_RETRY. Therefore it will require a connection point that identical to the initial publishing e*Way.

The database tables used for a publishing Error Hospital are identical and can be the same actual database as a subscribing Error Hospital. The same deployment issues exist.

SeeBeyond tools

This section provides a brief overview of SeeBeyond administration tools. Additional information about the SeeBeyond tool set may be found in the SeeBeyond documentation.

e*Gate Monitor and JMS administration tools

The main tool used for starting or stopping a system is the e*Gate Monitor application. This application attaches to a control broker and is designed to manually start, stop, pause, resume, or retrieve the status of a component.

The e*Gate monitor is a GUI that can display all components found in a specific schema. Additional GUI applications are accessible from the e*Gate monitor. There is a queue monitor for SeeBeyond standard JMS queues called the JMS Administrator.

The queue monitor tools allow an administrator to examine the number of messages on a queue and to view the contents of a message on a queue.

Details about the e*Gate Monitor application is found in the SeeBeyond e*Gate Integrator System Administration and Operations Guide. Details about the JMS Administrator application are found in the SeeBeyond JMS Intelligent Queue User's Guide.

e*Gate enterprise manager

The e*Gate is an application that is used for e*Gate development and operational changes. It is the primary tool for operations personnel for defining the EAI system's security roles and defining new users.

Command line utilities

The following commands can be issued from a command line interpreter, such as the Korn Shell in Unix or a DOS window. These commands should be found in the directory <EHOME>/client/bin, where <EHOME> is where the e*Gate software was installed. Many commands also require shared libraries or DLLs. On Unix systems, the directory <EHOME>/client/bin may need to be inserted into the LD_LIBRARY_PATH variable.

On Unix systems each command has the form <command> or <command>.exe. Only the latter form is executable on Windows platforms.

stcinstd

This command is known as the "Installer Service". This service is used to register a host name with the registry as a valid EAI participating host. This command performs two functions:

1 It allows users to edit the host and domain name properties for a participating host in the e*Gate Enterprise Manager application

2 It enables the e*Gate system to automatically propagate upgrades made to a Registry host to all participating hosts.

The `stcinstd` command should be run at least once per participating host so that the host name can be registered.

stcregutil

This is a command designed to modify, import, export or display information on an existing registry. A common usage will be for importing or exporting e*Gate schema information from development, test, and production environments. It does allow fine-grain control over the import and export process. Much of this functionality is also part of the e*Gate Enterprise Manager tool. However, this utility may be a large asset when defining code migration procedures for new EAI system releases.

stcaclutil

This is a utility used to define Access Control List (ACL) privileges, roles, and user properties. These functions may also be performed using the e*Gate Enterprise Manager application. Privileges can be assigned to roles and users assigned to roles. Users and roles can be added or deleted. User passwords may be altered.

stciqutil

This is a utility for manipulating the contents of a SeeBeyond standard Intelligent Queue. However, this is of a limited utility for RIB components, since the RIB uses SeeBeyond JMS Intelligent Queues.

stcutil

This is a utility designed for system testing and debugging. It is of limited use when working with RIB components.

stccmd

This is a text-based version of the e*Gate system monitoring tool. As such, it duplicates much of the functionality found in the e*Gate Monitor application. It provides a command line interface for status retrieval and component starting, stopping, and status retrieval. It may also “resolve” alerts. Available commands include:

```
? - list available commands
activate <component name> - activate element operations
attachiq <IQ name> - IQ to bring up
cls [cmd|stat] - clear window
debug <component name> [flag] - show or change an element's debug
flags
detachiq <IQ name> - IQ to detach
exit - exit stccmd.exe
getres [-b<begin date (mm/dd/ccyy)> | -e<end date (mm/dd/ccyy)] -
show resolved notifications
getstatus [-b<begin date (mm/dd/ccyy)> | -e<end date (mm/dd/ccyy)] -
show status-type notifications
getunres [-all | -a] - show unresolved notifications
help <command> - on-line help
history - list command history
list  [
all | monitors {-m} | alertors {-a} | iq {-i} | control {-c}
| notif {-n} [flush | all
| -b<begin  date (mm/dd/ccyy)>  [-e<end date (mm/dd/ccyy)>]
| +r | -r | -i<notification number> | <component name>
]
 quit - exit stccmd.exe
 reload <component name> [hard] - reload configuration
 resolve <notification number> - indicate that a notification has
been resolved
 sequence <component name> [value] - show or change sequence number
 shell <shell command> - run an external command
 shutdown <component name> - controlled module shutdown
 shutdownall <shutdownall> - controlled modules shutdown
 start <component name> - start or restart module
 startall <startall> - start or restart all modules
 status <component name> - show status
 suspend <component name> - suspend operations
 version <component name> - Show version
```

As with the e*Gate Monitor, not all commands are appropriate to all components.

The `stccmd` command may be used interactively or as a line in a shell script. For example, to list all component statuses, issue the command:

```
stccmd.exe -rh egate_main -rs RIB102 -cb egate_cb -un Administrator
-up STC -cmd list all
```

Where `egate_main` is the registry host, RIB102 is the schema name, `egate_cb` is the control broker to connect to, `Administrator` is the e*Gate user name to use, and `STC` is the password for the Administrator user.

stcmsctrlutil

This utility is used to examine and manipulate a JMS IQ Manager configuration and current messages. The command line format is:

stcmsctrlutil –host <hostname> -port <tcp port> <<COMMAND>>

where

<hostname> is the name of the host hosting the JMS IQ Manager

<tcp port> is the port number of the JMS IQ Manager

<<COMMAND>> is one of the legal commands for the stcmsctrlutil program. Useful commands are:

`-topiclist`
lists all defined topics

`-topicstat <topic name>`
lists statistics for the named topic.

`-sublistfortopic <topic name>`
lists all subscribers defined for a topic

`-createtopic <topic name>`
creates a new topic

`-deletetopic <topic name>`
deletes an existing topic

`-createsub <topic name> <sub name> <client name>`
creates a new durable subscriber for the topic with the given subscriber name and client name

`-deletesub <topic name> <sub name> <client name>`
deletes an existing durable subscriber for the topic with the given subscriber name and client name

`-tmsglist <topic name> <starting seqNo> <# of msgs>`
Displays the messages found in the named list

`-tmessage <topic name> <seqNo>`
Displays the contents of a single message

`-deltmsg <topic name> <seqNo>`
Deletes a message from a topic

RIB component configuration

This section details configuration issues and options with the RIB.

Oracle database triggers

Before any message can be published, a trigger may need to be enabled within the publishing application. Information on these triggers may be found in the RMS, RDM, or RCOM operations guides and reference manuals.

RIB property file

The RIB property (rib.properties) file uses the standard Java property file format. It specifies Error Hospital, TAFR, logging and other configuration information.

- For specific entries dealing with the Error Hospital, see the Message Error Hospital chapter.
- For specific entries dealing with TAFR adapters, see the TAFR Configuration section detailed later in this chapter.

The RIB properties file must have the name “rib.properties”. However, the location of this file may be specific to the e*Way using it.

Multichannel_ind property

The only other type of RIB property file entry is used by RMS publishers. It is the “multichannel_ind" property. An example of an entry here is

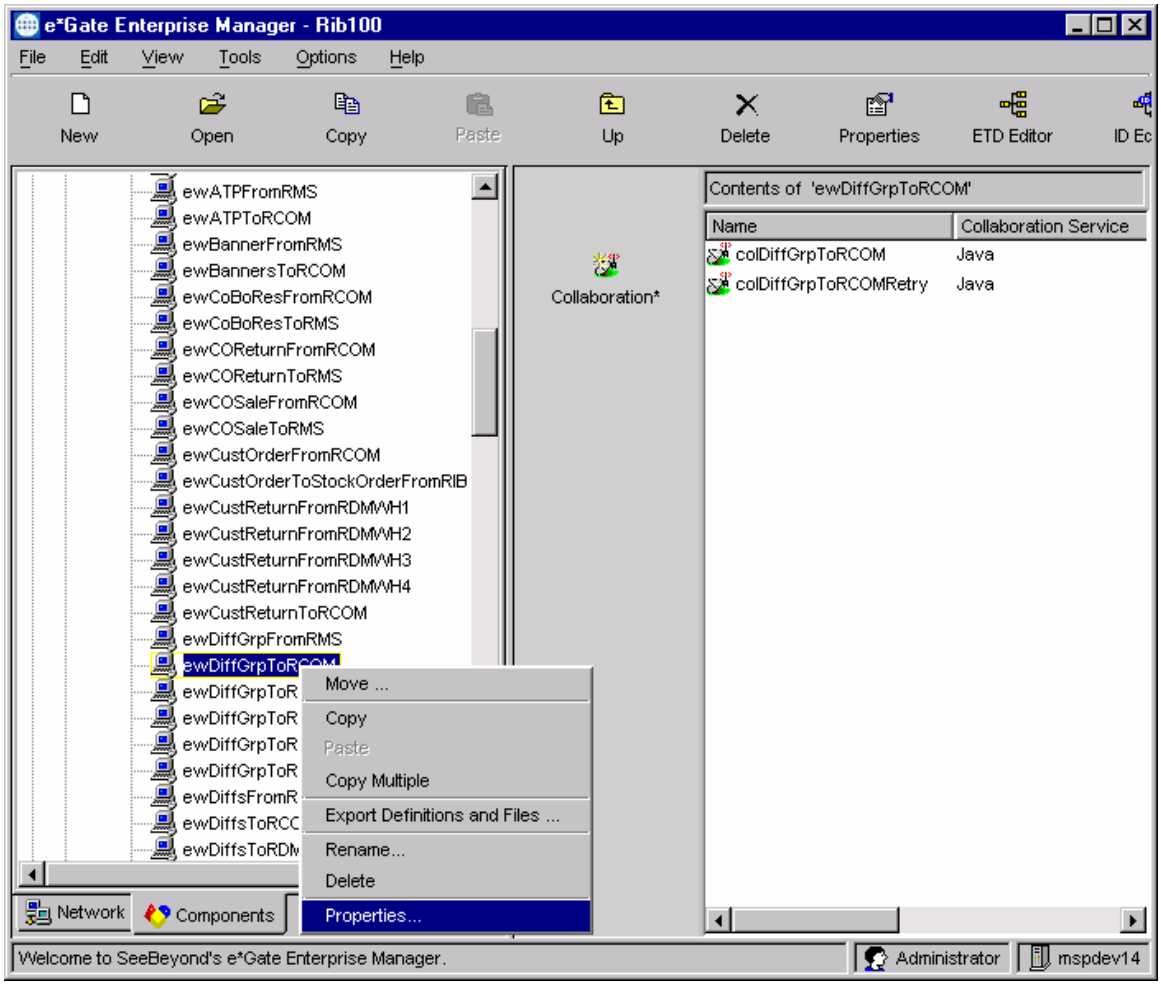
```
multichannel_ind = MPHYS
```

Valid values for this property are:

- MPHYS Specifies multi-channels using physical warehouses. The effect is for RMS to consolidate virtual warehouse orders at a physical level.
- S Specifies a single distribution channel is in use.
- M (Reserved for future use).

SeeBeyond e*Way configuration files

All RIB adapters are SeeBeyond Multimode e*Ways. Each uses its own configuration file containing parameters it needs to function. These configuration files can be manipulated by the SeeBeyond e*Gate Enterprise Manager application.

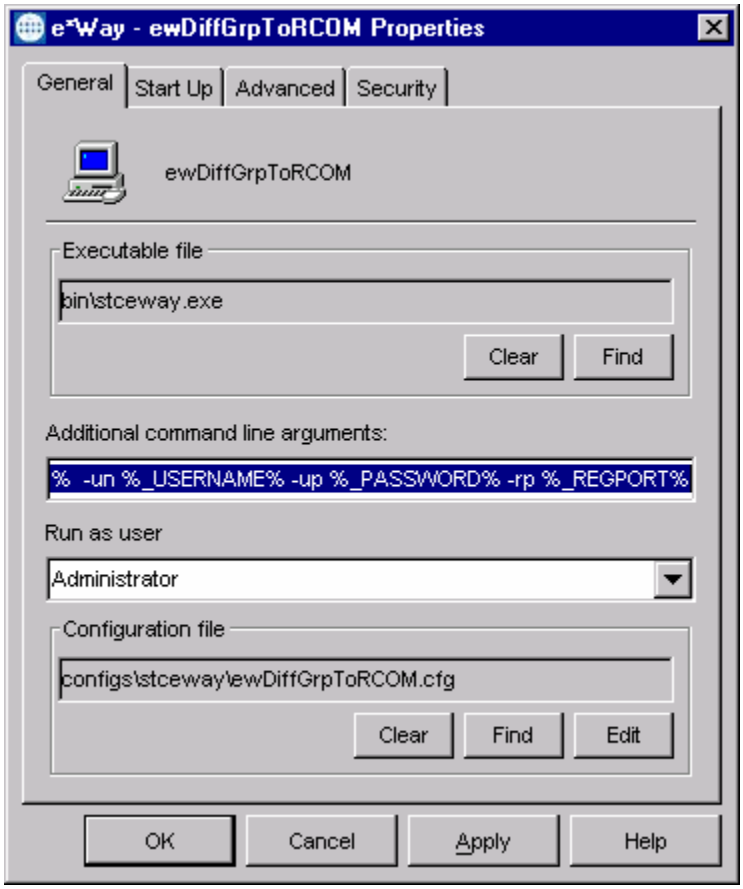


Right-click on e*Way in e*Gate Enterprise Manager

e*Way property and configuration files

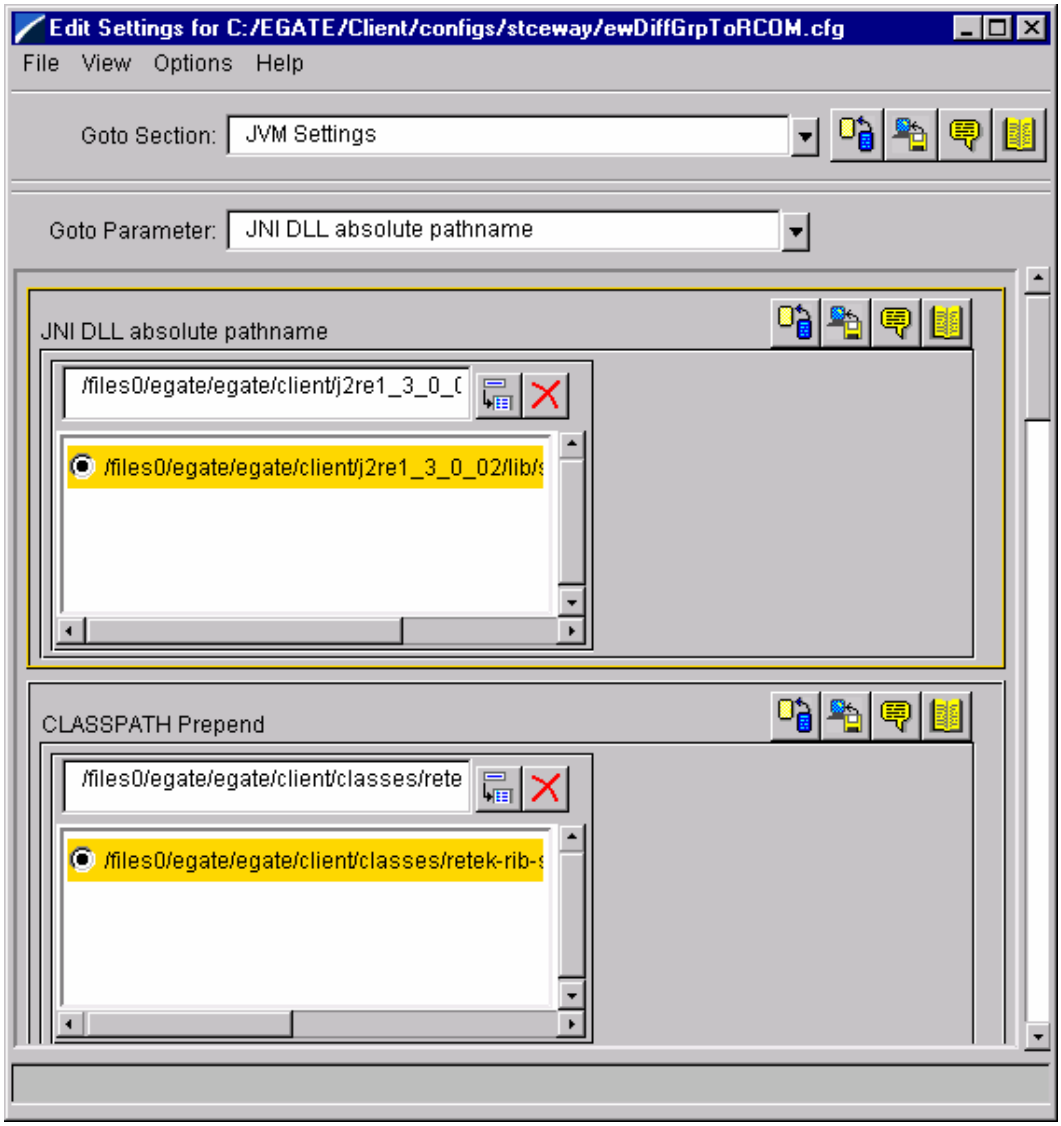
The following shows what is displayed when you right click to select an e*Way, to modify its properties.

1 Select Properties… from the menu, or click the Properties toolbar icon. The e*Way Properties dialog box is displayed.



e*Way Properties Window

2 Click **Edit**. The Configuration Edit window is displayed.



e*Way Configuration Edit Window

The configuration for this e*Way is the file <EHOME>\configs\stceway\ewDiffGrpToRCOM.cfg.

3 Verify the main configuration entries:

- JNI DLL absolute pathname

  The JNI DLL absolute pathname is the location of the Java Native Interface library. On Unix systems, this is a shared library, while on Microsoft Windows platforms this is a DLL. This library provides access to native 'C' language components that are part of the SeeBeyond e*Way infrastructure. SeeBeyond provides such a library with its installation on a specific platform.

The name of the file on Unix systems is typically of the form "libjvm.so". On Windows it is "jvm.dll". From the SeeBeyond installation disk, this library is typically found under a Java Runtime Environment directory. Examples of the library's location include:

```
<EHOME>\client\Jre\1.3\bin\hotspot\jvm.dll
(Microsoft Windows)
```

```
<EHOME>/client/j2re1_3_0_02/lib/sparc/client/libjvm.s
o
```

(Sun SunOS or Unix)

- CLASSPATH Prepend

  The "CLASSPATH Prepend" parameter must include the location of the RIB class Java Archive (JAR) file and the location of the RIB properties file. Both the RIB Support JAR and the rib.properties file are typically found at

  ```
  <EHOME>/client/classes
  ```

  Hence, an example of the CLASSPATH Prepend parameter on a Unix system is (assuming e*Gate is installed in EHOME (/opt/egate))

  ```
  %_EHOME_%/client/classes
  ```

  while, if e*Gate is installed in C:\egate on a Microsoft Windows system:

  ```
  %_EHOME_%\client\classes
  ```

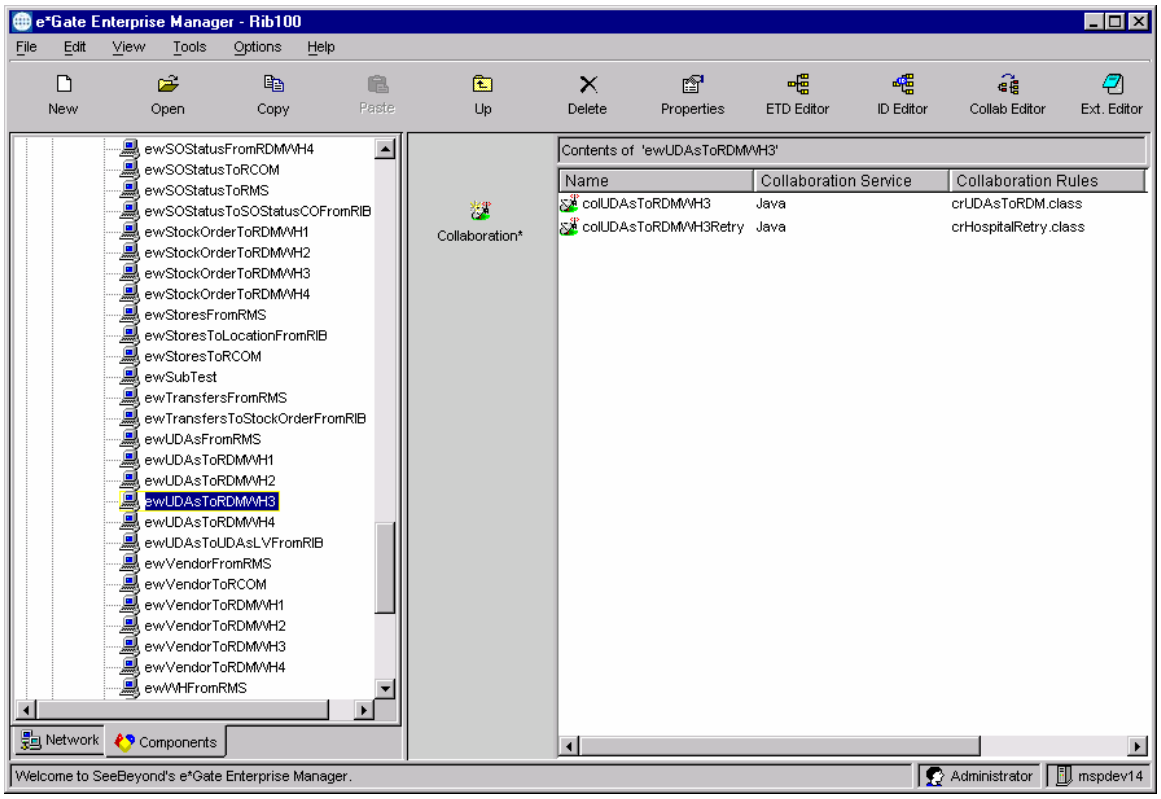
Note: The path separator is a semi-colon on the Windows system, and a colon on the Unix system.

e*Way collaborations

Collaborations define the processing logic for a message. They also define where messages are subscribed from and published to. For many e*Ways, there will be no need to modify the collaborations specified for an e*Way. This is because the supplied connection points can be modified for site-specific values, such as the host name or TCP port.

However, modifications to the Collaborations specified in an e*Way are needed when new connection points are required. An example of this is for a new RDM installation in a remote warehouse. The RDM instance will have its own database and therefore a new Oracle Connection Point is required. An additional Error Hospital for such an installation may be useful for performance reasons. The remote installation may also require a local JMS IQ Manager and associated connection point. It is possible to have three or more additional connection points per new RDM installation. This is in addition to creating the new remote participating host.

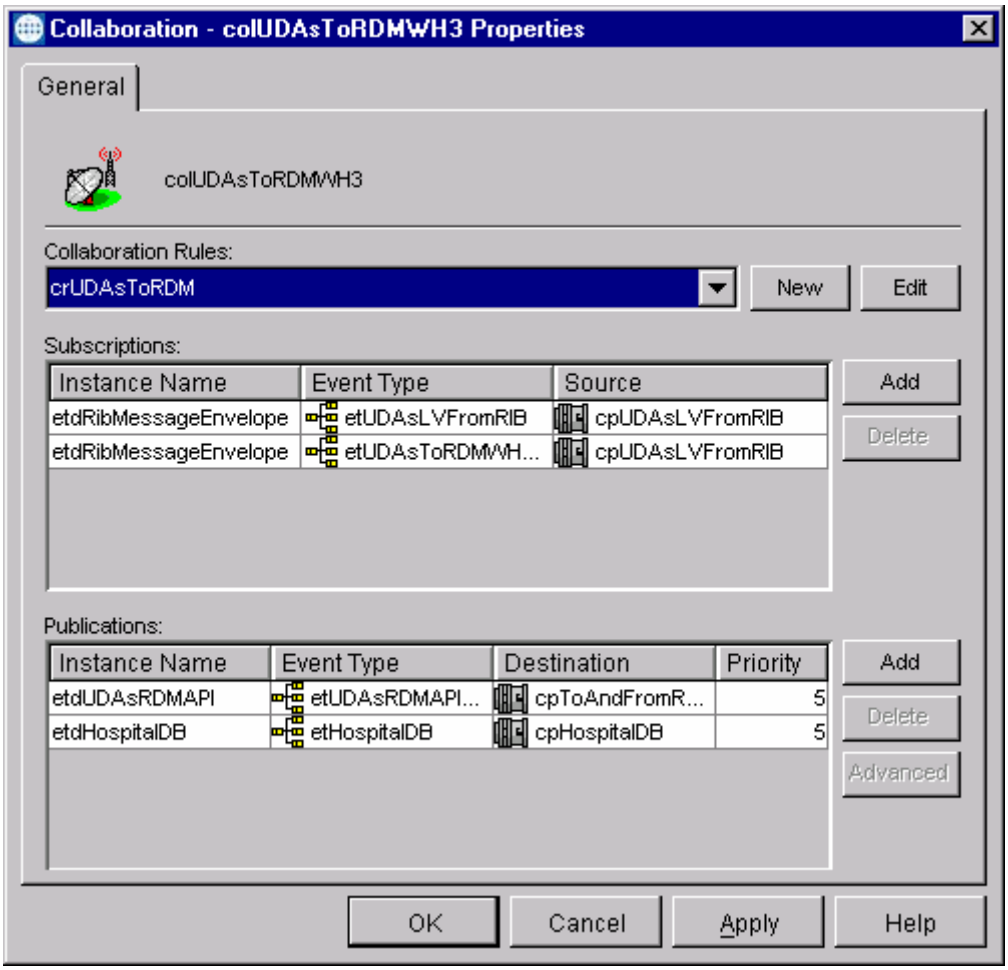
The figure below shows the main e*Gate Enterprise Manager for a RIB adapter.



Main e*Gate window when RIB e*Way selected

The e*Way selected is a subscribing interface to RDM for one warehouse (number 3 out of 4). The collaboration colUDAsToRDMWH3 subscribes to the UDA message family and is the normal “subscribing” collaboration. The collaboration named colUDAsToRDMWH3Retry is the “retry” collaboration and is responsible for resubmitting and deleting messages from the Error Hospital for the UDA message family for this subscriber.

When the properties of colUDAsToRDMWH3 are examined, the following window is displayed:



Subscribing e*Way collaboration properties

There are two Event Types subscribed to in this example: One for unprocessed messages (etUDAAsLVFromRIB) and one for messages to be re-processed (etUDAsToRDMWH3Retry). The source for each type is the connection point cpUDAsLVFromRIB.

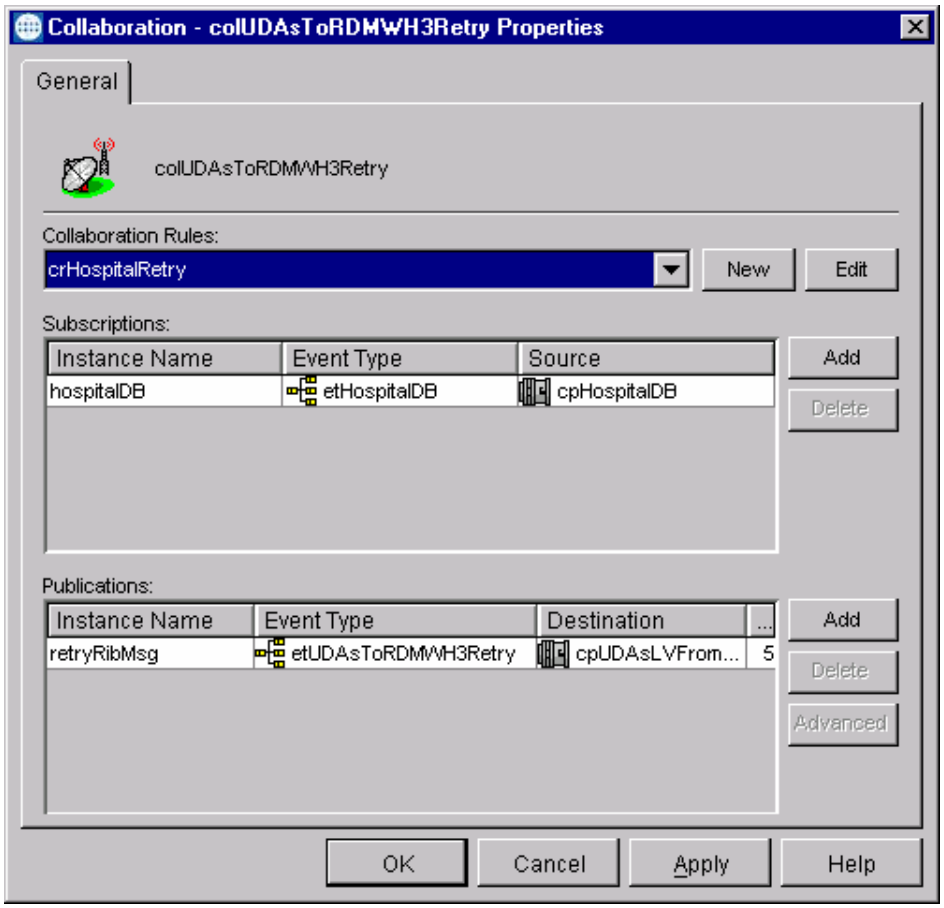
Note: This example uses a single JMS queue for all e*Ways in the EAI system. If a local queue were used, the connection point should be named something similar to cpUDAsLVFromRIBWH3.

There are also two Event Types “published” in this example: `etUDAsRDMAPIWH3`, the Oracle connection point associated with the warehouse specific RDM instance and `etHospitalDB`, the Error Hospital Oracle Connection Point.

Note: This example uses a single Error Hospital for all e*Ways in the EAI system. If a local Error Hospital were used, the connection point should be named something similar to cpHospitalDBWH3.

Note: This is a subscribing collaboration; the “publishing” connection points serve only to provide the database connection within the processing logic. No messages are published to any queues for this collaboration.

However, the “retry” collaboration does publish messages to a queue. The retry collaboration’s properties is seen below:



Retry collaboration properties

For the retry collaboration, the subscription “source” is the Error Hospital Oracle Connection Point, not a JMS queue. For publishing messages, the retry collaboration uses the same connection point as the subscribing collaboration. The event type it publishes is the etUDAsToRDMWH3Retry event.

If the retry collaboration published the same event type that the subscribing collaboration originally processed (and had a problem with), then *all* subscribers to this event type would re-process the message. In this particular case, this would not be a problem, since this event type only has one subscriber. However, other event types are subscribed to by multiple applications. Problems can arise when a message is delivered after it has been processed successfully.

SeeBeyond connection point configurations

All RIB Adapters use connection points as a source/sink for messages and for accessing databases. This section details the configurations for the JMS Connection Point and an Oracle Connection Point.

The most important aspect of this configuration is the use of the XA protocol in support of processing messages exactly once.

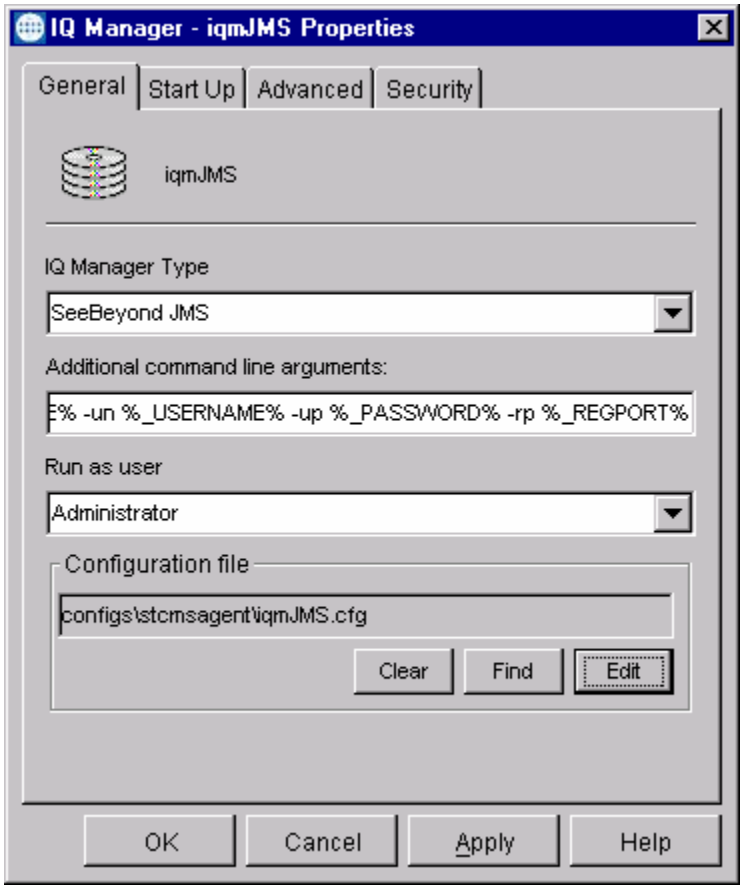
JMS IQ manager configuration

Configuring a JMS connection point requires knowledge of the Java Message Service server that is to be used. SeeBeyond's JMS Intelligent Queue Manager provides such a service. Other message oriented middleware products, such as IBM's MQ Series product, also may provide such services.

A JMS server provides access to one or more JMS Queues and their associated stable (a.k.a. hard disk) storage. Multiple JMS IQ Managers may be created and deployed with the RIB, depending on the topology of the installation, message lifecycle, administration, performance and availability requirements.

Although a JMS IQ Manager may be accessed from multiple e*Gate schemas via the connection points contained in these schemas, only the schema containing the JMS IQ Manager can administratively view the messages contained in the JMS server queues.

Similar to other e*Gate components, the JMS IQ Manager's full operating parameters are found in two windows: An IQ Manager Properties window and the JMS IQ Manager specific configuration edit window.



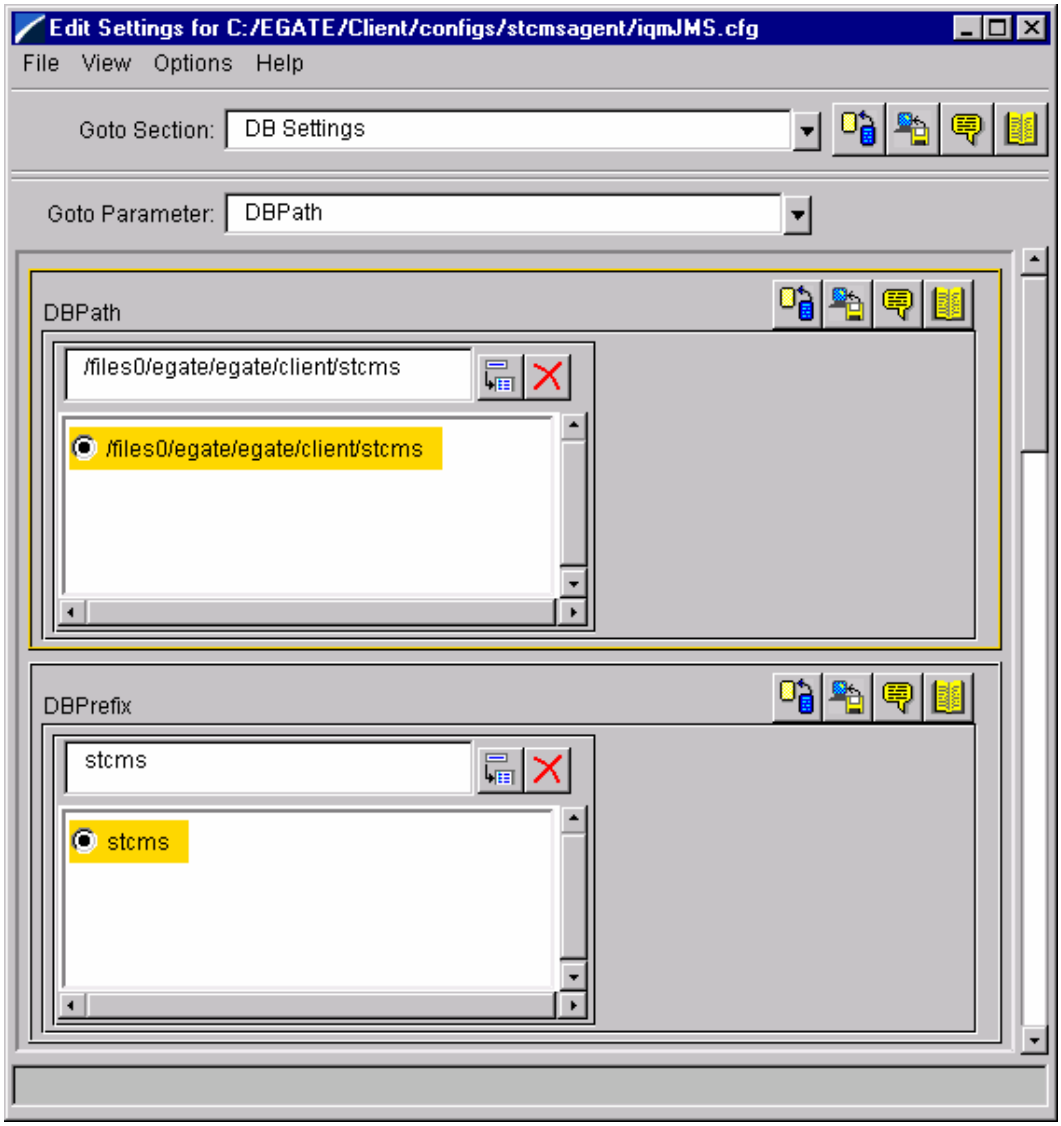
JMS IQ Manager Properties Window
The following properties are extremely important:

- On the "General" Tab:
  - IQ Manager Type: By definition, must be SeeBeyond JMS.

    Note: Of course, if an enterprise has standardized on the IBM MQ Series product for JMS servers, then the SeeBeyond MQ Series Connection Point will be used directly with this server. In this case, no JMS IQ Manager is needed.

  - Configuration File: Details IQ manager configuration storage.
- On the "Start Up" tab:
  - Start Automatically: determines if the IQ Manager's control broker will start up the IQ Manager whenever the control broker starts up.
- On the "Advanced" Tab:
  - TCP/IP port number: determines the TCP port number to listen on. This must be allocated specifically to the JMS IQ manager instance. No other application (including other JMS IQ Managers) can use this port.
  - Log: This button accesses an additional window to control logging and tracing levels.

- On the “Security” Tab:
  - Privilege: Allows access to a window assigning privileges to defined roles when ACL’s have been enabled.



JMS IQ Manager Configuration Edit window

The SeeBeyond e*Gate JMS IQ Manager configuration contains five sections. Full documentation on these parameters is found in the SeeBeyond JMS Intelligent Queue User's Guide.

1 **DB Settings:** This section defines the stable storage options for the files used by the JMS server. The "DBPath" configuration parameter is particularly interesting, since it locates the file directories used to store messages. It also provides options for disk synchronization and memory cache size.

Note: If left blank, the value of the MessageServiceData property from the .egate.store file will be used. This file is normally located in the user's home directory.

2 **Message Settings:** This section specifies options for allocating memory for messages and the maximum time a message will be allowed to persist on a queue within the server.

3 **Server Settings:** This section defines the maximum number of messages the server will store. The JMS server will throttle clients (cause them to wait) when this number is exceeded.

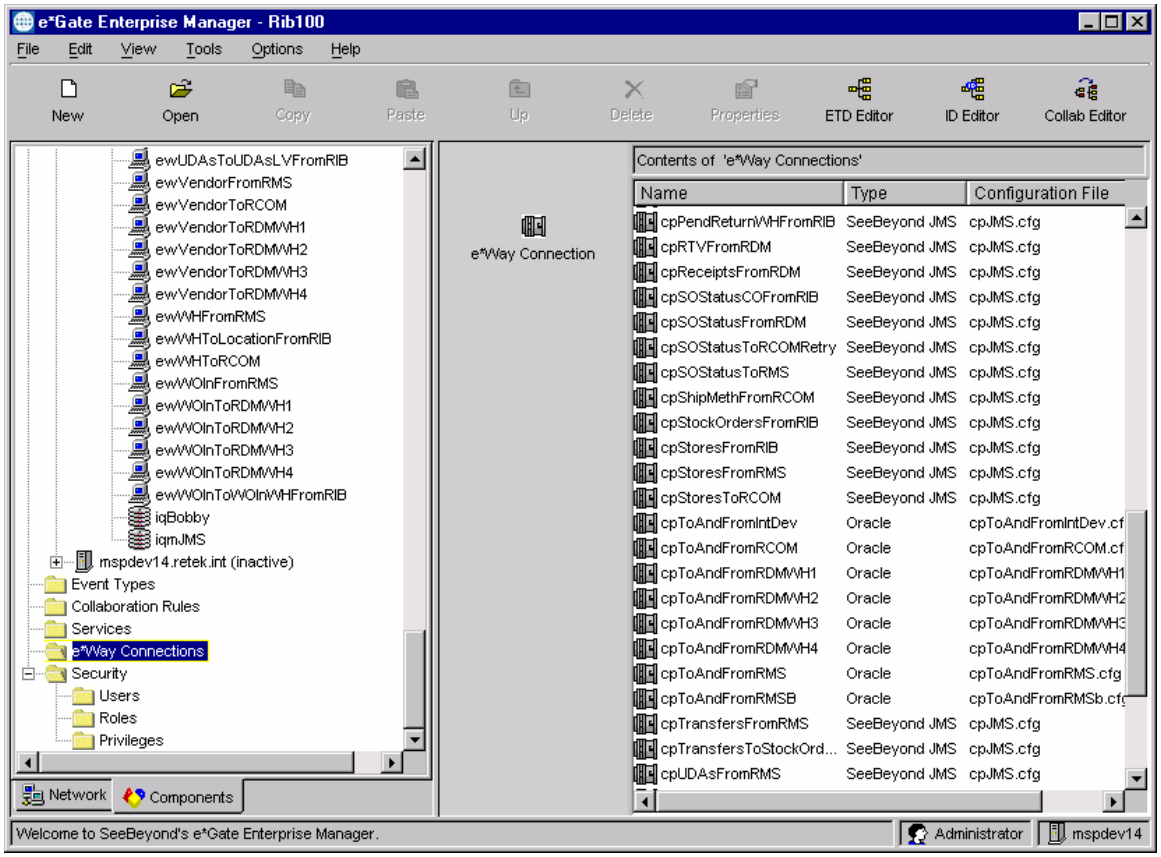
4 **Topic Settings:** This section sets the per-topic resource limits. In the RIB environment, a topic equates to an e*Gate Event Type which equates to a specific queue of messages supplying a set of subscribers.

5 **Trace Settings:** This section controls tracing of messages for the JMS server. Parameters include the name of the log file used for tracing, the trace verbosity level, and specific types of tracing to perform.

Note: Remember that configuration changes need to be promoted to the run time environment before they take effect. To do this: in the Configuration Edit window, select File > Promote to Run Time.

JMS IQ Connection Point configuration

JMS Connection Points are defined within the e*Way Connections folder. This folder is found at the right-hand e*Gate Enterprise Monitor frame near the bottom. When selected, the window will appear similar to the figure below:



e*Gate Enterprise Manager with e*Way Connections folder selected

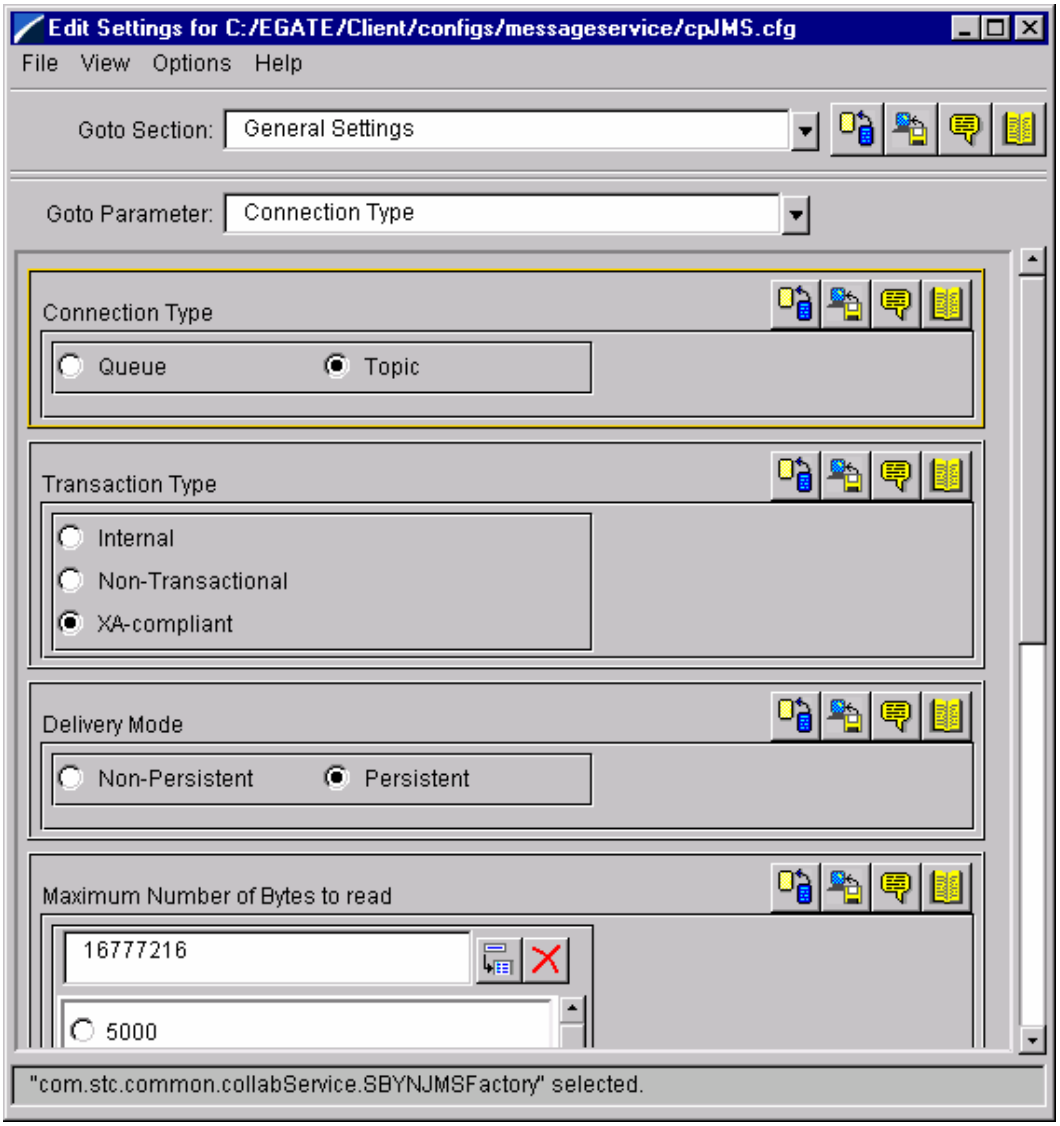
To create new connection points:

- Click the central e*Way connection button.

To edit existing connection points:

1 Select the connection point.

2 Modify the connection point's properties: the two main properties are the configuration file and the connection point type (which by definition must be a SeeBeyond JMS Connection Point).



JMS Connection Point Configuration Edit window

There are two sections determining the connection point's operating characteristics:

- **General Settings:** This section details standard JMS operation options and message restrictions for the JMS client. Parameters for the General Settings include:
- **Connection Type:** Specifies if the connection type used is as a "Queue" or a "Topic". Must be set to "Topic" to ensure that all subscribers get the message. When "Topic" is specified, all subscribers will receive a copy of all messages for all queues managed by the JMS provider. If "Queue" is specified, then no message will be sent to more than one subscriber and the allocation messages to subscribers is indeterminate.
- **Transaction Type:** Specifies the type of transactions used to dequeue and enqueue messages. "XA-Compliant" must be used for messages to guarantee messages are processed successfully exactly once within the RIB.
- **Delivery Mode:** Must be set to "Persistent" to insure messages are written to disk before an enqueue operation completes.
- **Maximum Number of Bytes to Read**: Specifies the maximum number of bytes to read at a single time from the received bytes message.
- **Default Outgoing Message Type:** The JMS standard specifies two types of messages: one consisting of bytes and one of strings. This is not to be confused with the RIB "message type".
- **Factory Class Name:** Name of factory class to use in creating the JMS connections. Suggested value:

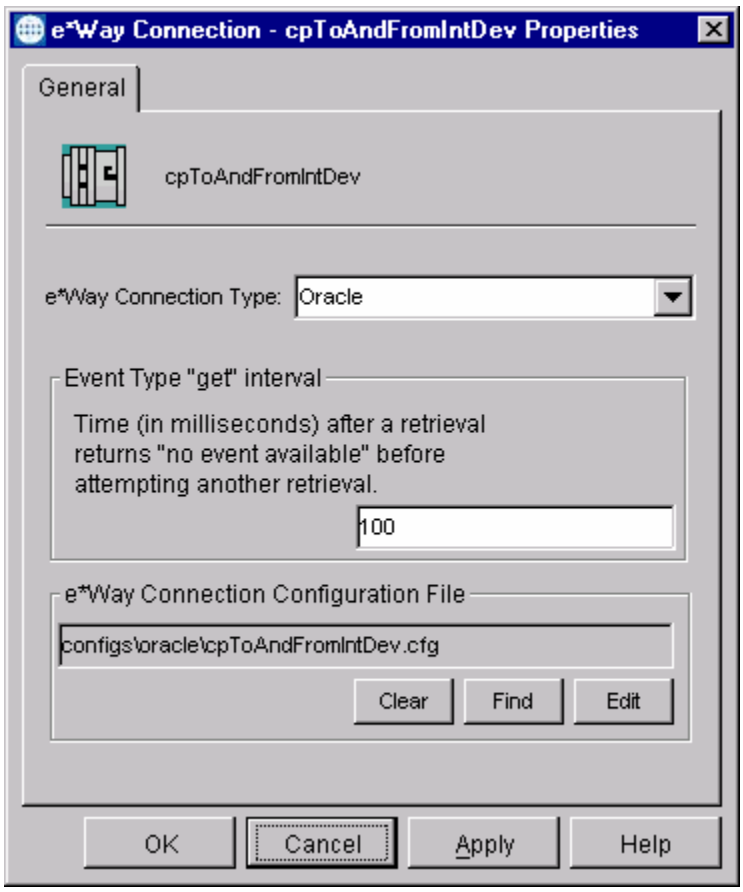
  `com.stc.common.collabService.SBYNJMSFactory`
- **Message Service:** This section details JMS IQ Manager specific parameters for the JMS server.
- **Server Name:** Specifies the JMS IQ Manager name as seen in the e*Gate Enterprise Manager application.
- **Host Name:** Specifies the IP address or the host name from a Domain Name Server (DNS) that is running the JMS IQ Manager.
- **Port Number:** Specifies the TCP Port number the JMS IQ Manager is listening on. Must match the JMS IQ Manager "TCP/IP Port Number" property.
- **Maximum Message Cache size:** Specifies the maximum message cache size for the connection point.

Note: Remember that configuration changes need to be promoted to the run time environment before they take effect. To do this, on the Configuration Edit window, select File > Promote to Run Time.

Oracle Connection Point configuration

Oracle Connection Points are defined within the e*Way Connections folder. This folder is found at the right-hand e*Gate Enterprise Monitor frame near the bottom. When selected, the window that is displayed is similar to Figure 7-9: e*Gate Enterprise Manager with e*Way Connections folder selected.

When the properties window of an Oracle Connection Point has been selected, it appears similar to the figure below:

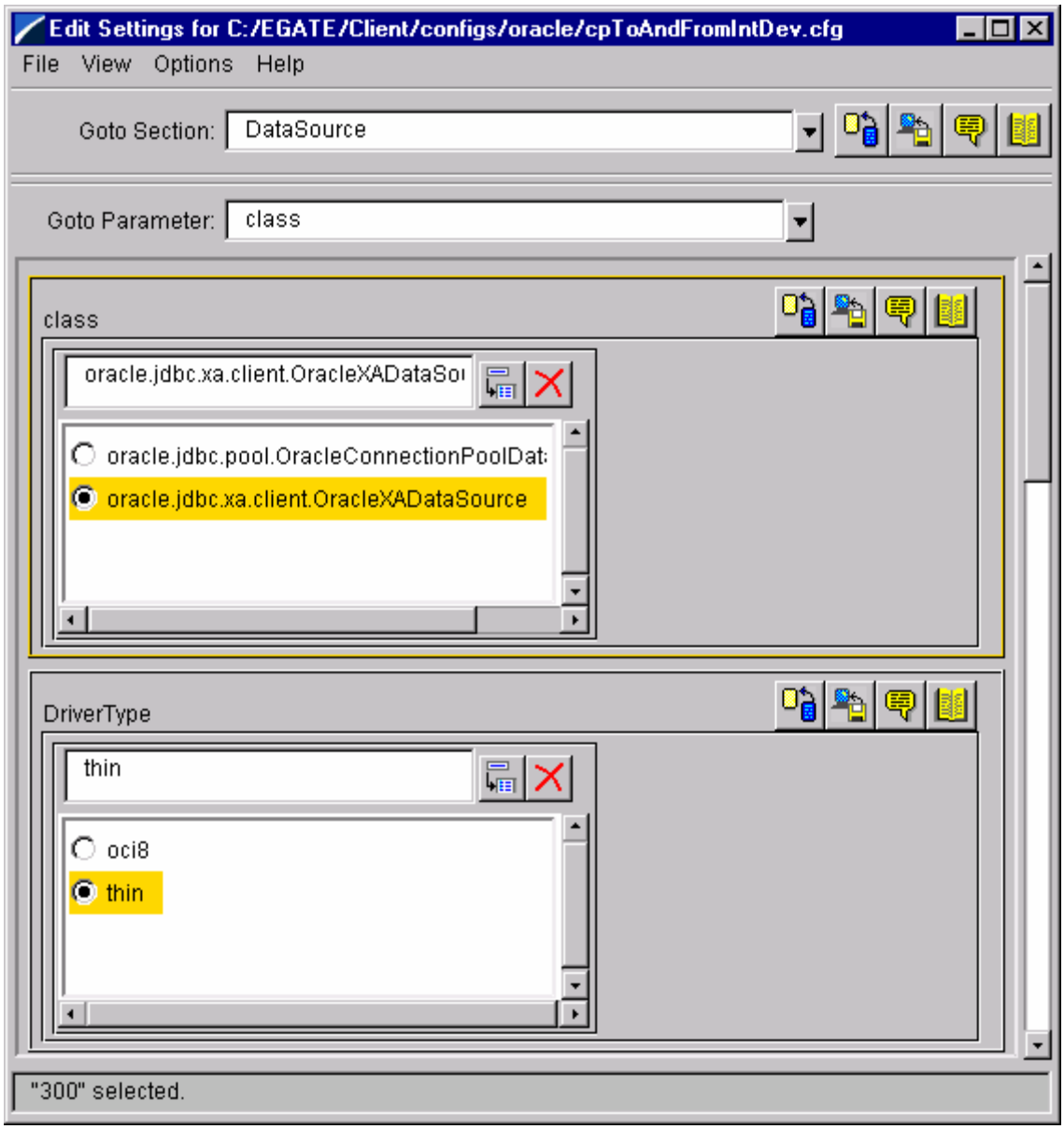


Oracle Connection Point Properties window

The properties are:

- **e*Way Connection Type:** Oracle, by definition
- **Event Type "get" interval:** This is a polling interval occurring after an "empty" data retrieval. Increasing this value may reduce load on a system. Decreasing this value may reduce the time it takes to publish a message by the RIB.
- **e*Way Connection Configuration File:** name of the configuration file storing additional parameters.

An Oracle Connection Point Configuration Edit window is pictured below:



Oracle Connection Point Edit window

There are two sections found in this configuration: “DataSource” and “connector”. The connector section contains two parameters that cannot be changed. The DataSource contains the following parameters:

- **class:** Specifies the JDBC driver class. For XA support, the class should be oracle.jdbc.xa.client.OracleXADataSource. The JAR file containing this class is typically found in <ORACLE_HOME>/jdbc/lib/classes12.jar.
- **DriverType:** Type of driver. The OracleXADataSource is a “thin” driver.
- **ServerName:** Name of the host containing the Oracle Listener process to connect to.
- **PortNumber:** TCP Port number the Oracle Listener uses to listen on for new connections.
- **DatabaseName:** System ID (SID) of the database to connect to.
- **UserName:** User name to use for the database connection.
- **Password:** Password corresponding to the user name. Stored as an encrypted string.
- **Timeout:** Login timeout value. Longest time to wait for a session to be established with the database.

Note: Remember that configuration changes need to be promoted to the run time environment before they take effect. To do this, on the Configuration Edit window, select File > Promote to Run Time.

Oracle Schema owner issues to consider

The Oracle connection point’s user name also depicts the schema name in which the Oracle connection will use by default. This user/schema is not required to be the owner of the package or the Database Objects being used. However, synonyms for all of the packages containing GETNXT() and CONSUME() must be present for the RIB user-id being used, and furthermore the owner of these packages containing the GETNXT() or CONSUME() stored procedure is required to be the owner of the RIB Objects as well.

The appropriate privileges for accessing the RIB Objects and executing the stored procedures must also be granted to the RIB user-id. Most often, the two privileges needed for a separate RIB user-id above those normally granted are 'CREATE ANY TYPE’ and 'EXECUTE ANY TYPE'.

TAFR adapter configuration

The TAFR adapter has both a SeeBeyond e*Gate configuration component and a RIB Properties file configuration component. Furthermore, when adding additional routing destinations, such as RDM warehouse installations, additional work must be performed.

RIB property file TAFR entries

The rib.properties file contains entries for an Error Hospital and for other components.

The properties associated with a TAFR are used to do the following:

- Translate facility ID codes to destination JMS queues and event IDs.
- Specify a default facility type when the publishing application has no knowledge of the facility type.

The entries in the rib.properties file for Facility ID translation have the following form:

```
facility_id.<FACILITY_TYPE>.<FACILITY_CODE> = <Dest>
```

where

`<FACILITY_TYPE>` is a string matching the available facility types for the entire set of locations.

`<FACILITY_CODE>` is a string matching the possible facility ID code values for a location.

`<Dest>` is a value to use for routing a message to a specific (warehouse) location. This will be appended to event type names to effect the routing of a message.

The entries in the rib.properties file for specifying the default facility type is

```
facility_type.default = <DEFAULT_FACILITY_TYPE>
```

This provides a means for translating messages created by publishers (such as RDM) that do not use the facility type abstraction.

TAFR Routing – adding new destinations

Transformation, Address Filtering/Routing (TAFR) adapters are designed to perform actions based on message content. Applications such as RDM require TAFRs to route messages to specific instances. The number and names associated with these instances are within the control of the implementation. This section details how to add or new destinations.

First, take a logical view of TAFR Processing. First, the message to be routed is published. The subscribing TAFR retrieves this message and, based on its content, re-publishes it zero or more times. The queues the TAFR uses to publish are different than the one it subscribes to.

The JMS IQ Manager the TAFR publishes to may be the same one it subscribes to, but the "topics" used to publish must differ – so that it will never subscribe to the same messages it publishes. Also, the SeeBeyond interface with the JMS IQ Manager equates a "topic" with an "Event Type". The RIB associates an "Event Type" to a "Message Family". A Message Family is a specific XML format. An Event Type is a tag applied to this format. Multiple Event Types may be associated with the same message family. Subscribers subscribe to messages with specific Event Types.

Note: The RIB associates an "Event Type" to a "Message Family". A Message Family is a specific XML format. An Event Type is a tag applied to this format. Multiple Event Types may be associated with the same message family.

When a TAFR determines the routing destination for a message, it uses a general-purpose API for publications. One of the parameters of this API is the topic to use. The TAFR computes the "topic" based on the destination and values in the rib.properties file. One risk with this design is that it is entirely possible for the TAFR to publish a message that has no subscribers. Another possible error is that the TAFR cannot compute the destination because of missing information from the rib.properties file. If either error is reported, then the TAFR will stop processing all further messages.

A summary of the steps used to add a new destination is as follows:

1 Determine which TAFR and Message Family requires routing.

2 Create the new Event Type name and definition.

3 Modify the TAFR's configuration to publish the new Event Type.

4 Create the destination messaging components.

Step 1: Determine which TAFR and Message Family requires routing

The first step in this process is to determine which messages are to be sent to the subscribing application. All message content information is found in the Retek 10.3 Integration Guide. This guide details the input and output event types for a TAFR processing the message family. In some cases, the documentation may picture multiple event types as input. The RIB schema as supplied from Retek deploys by default a separate TAFR adapter for each input event type.

Once the Message Family has been determined, the TAFR can easily be found, because the RIB uses the naming convention of:

```
ew<MsgFamily1>To<MsgFamily2><Dest>FromRIB
```

where

`<MsgFamily1>` and `<MsgFamily2>` are the names of message families used for input and output.

`<Dest>` is a generalized specification of the destination (for example, WH for RMD warehouses).

Step 2: Create the new Event Type Name and Definition

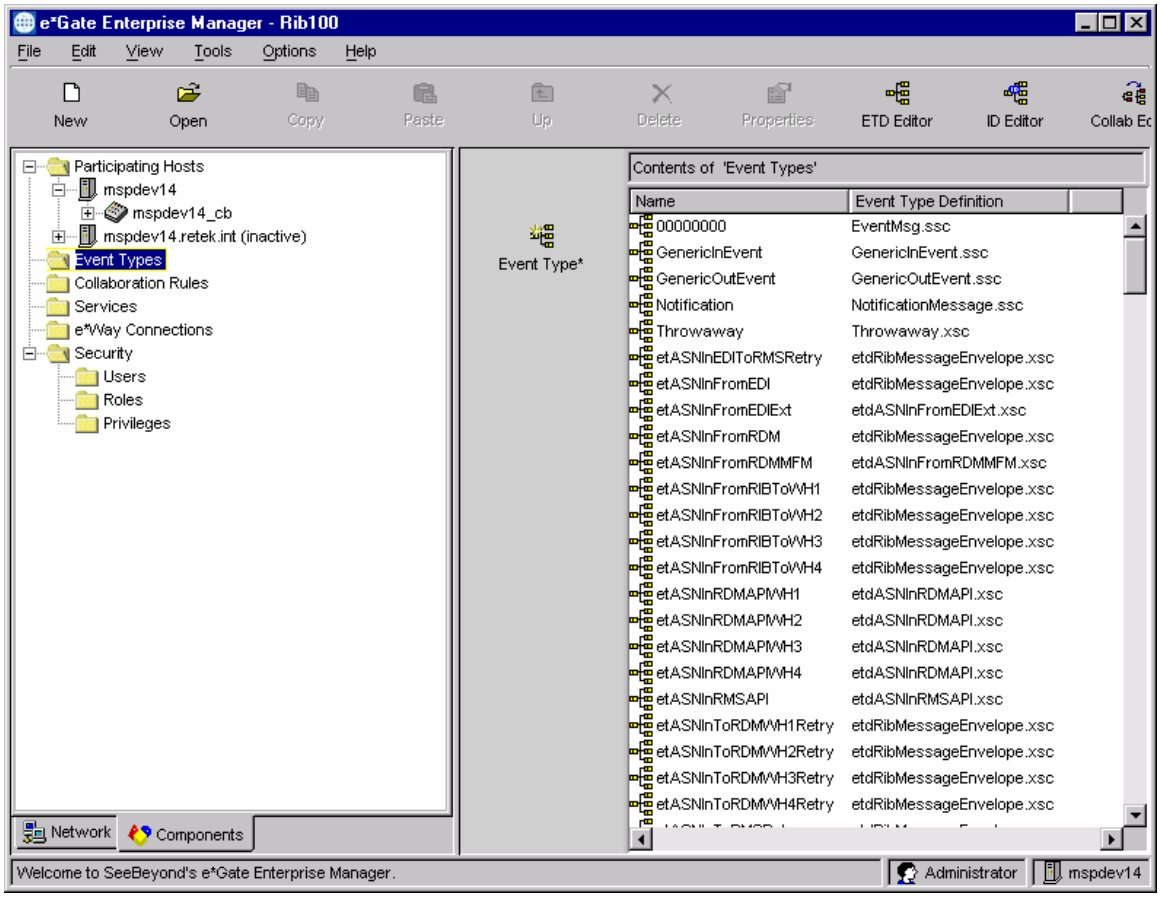
Two new event types will need to be created. The first is the new event type used by the TAFR component to route the message to the new destination. The second is used by the subscribing RIB adapter that interfaces with the application – the intended destination. These RIB e*Ways subscribe to two events, the “routed” message event type just mentioned and an event type associated with retrying the message if an error occurs.

The RIB uses the following naming convention for the Event Type names published by TAFR components:

```
et<MsgFamily>FromRIBto<DestSpec>
```

where <MsgFamily> is the message family name and <DestSpec> is the destination specification. An example is the Event Type name etASNInFromRIBToWH1. As mentioned above, the specific event types published is found in the Retek 10.3 Integration Guide.

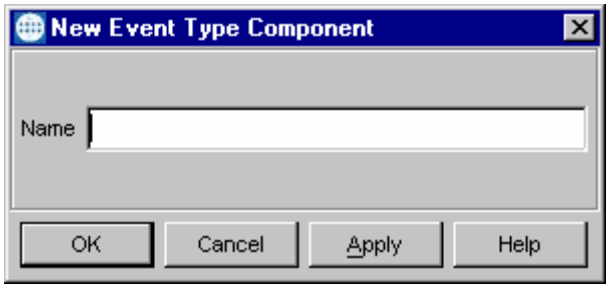
Once the name has been determined, the definition must be created. This is done via the e*Gate Enterprise Manager application. Clicking on the “Event Types” folder displays the following window:



e*Gate Enterprise Manager with Event Types folder selected

The figure above shows four possible published event types for the TAFRs involved with the ASNIn message family: etASNInFromRIBWH1, etASNInFromRIBWH2, etASNInFromRIBWH3, and etASNInFromRIBWH4.

Clicking on the central “Event Type*” button brings up the following window:



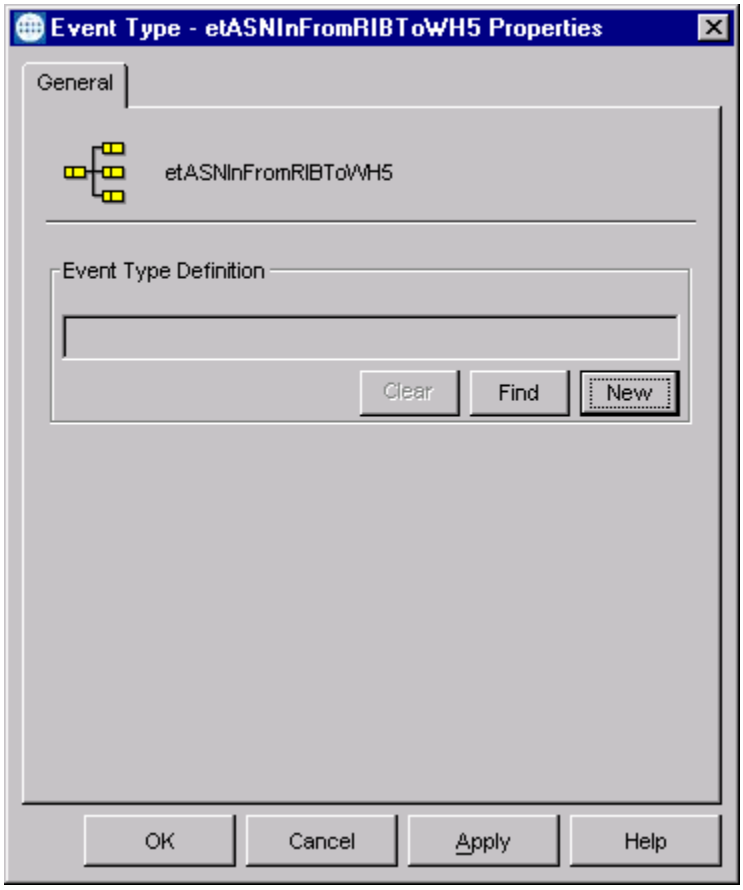
New Event Type window

1 In the Name field, enter the new event type name, for example, `etASNInFromRIBWH5`.

2 Click **OK**.

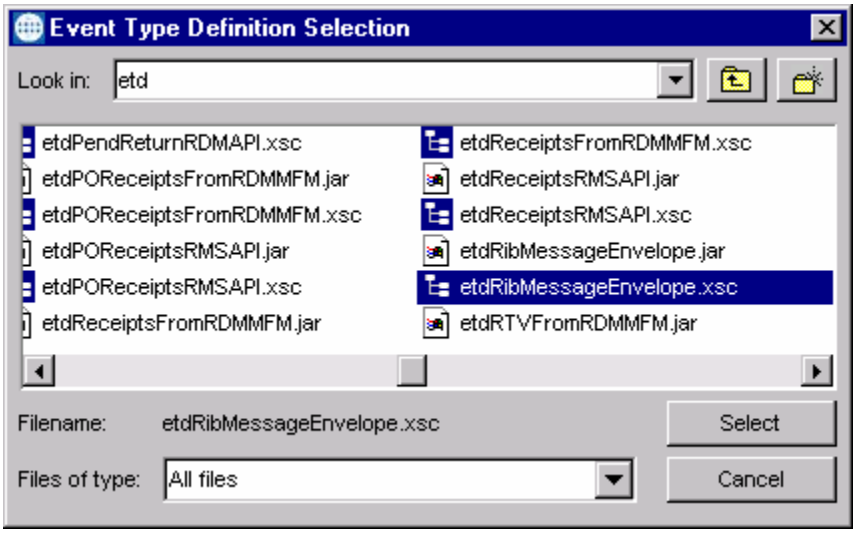
3 The new event type is displayed at the bottom of the list of event types.

4 Double-click on the new event type. The Properties window is displayed.



Event type properties window

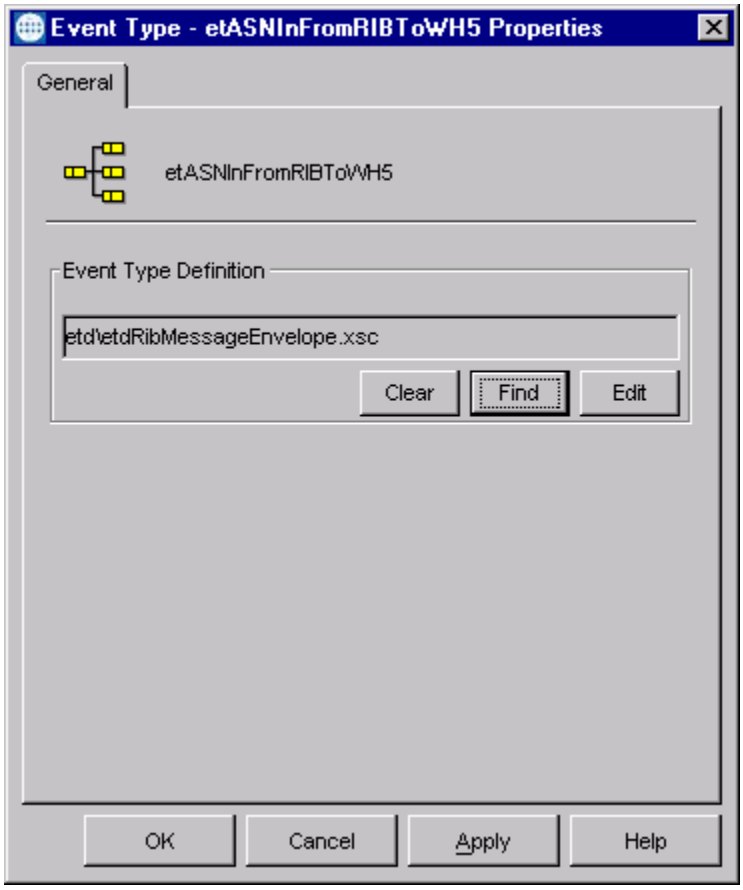
5 Click **Find**. This allows you to associate an existing message format (or Event Type Definition) with the new event type. (This may take a few seconds.) The Event Type Definition Selection window is displayed.



Choosing an Event Type Definition for the new Event Type

6 Select the etdRibMessageEnvelope.xsc file.

7 Click **Select**. The Event Type Properties window is displayed.



Updated Event Type Properties window

8 Click **OK** to finish creating the new Event Type.

Repeat this process for the “Retry” event type, using the following characteristics:

- The same Event Type Definition
- The Event Type Name of the form et<MsgFamily>To<DestSpec>Retry.

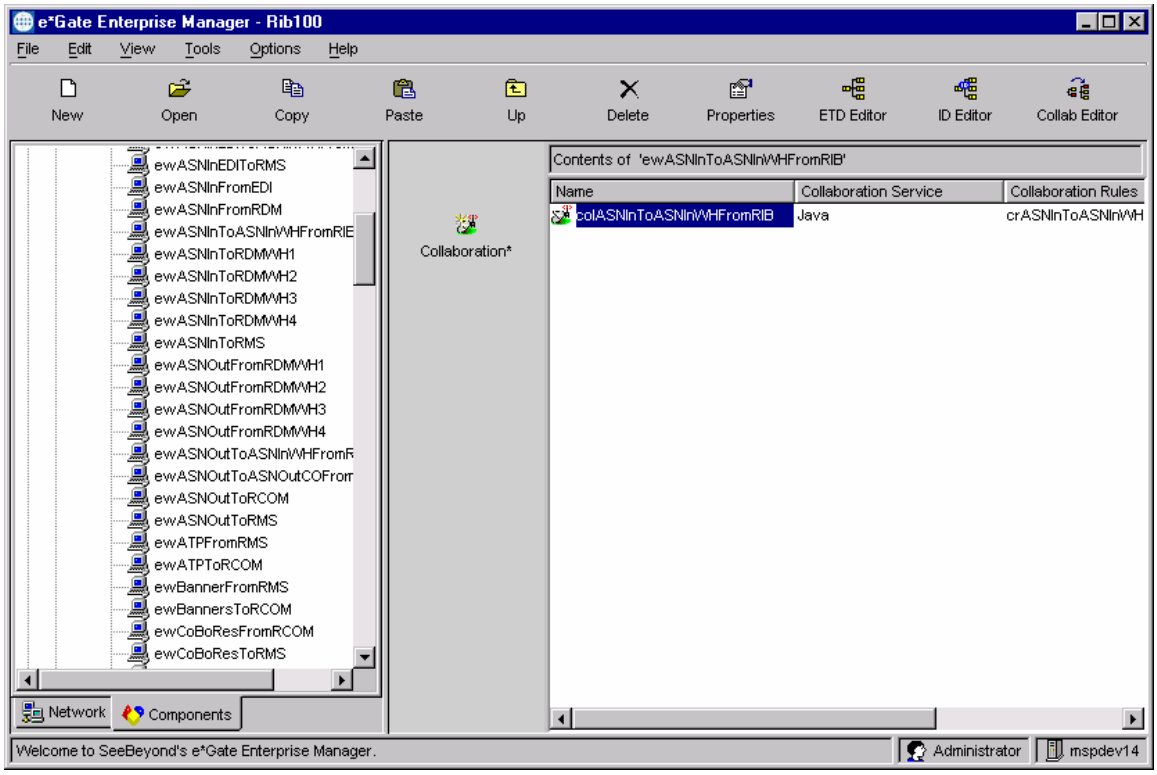
In the case of the examples above, the event type would be named etASNInFromRIBToWH5Retry.

Step 3: Modify the TAFR's Configuration to publish the new Event Types.

The next step is to publish the new event type. This has two parts: to update the e*Gate registry that the new event type will indeed be published, and, for messages destined for an RDM instance, modify the RIB properties file.

1 In the e*Gate Enterprise Manager, select the TAFR e*Way.

This can be a little tricky, since many names are similar. TAFR names have the form ew<MsgFamily>To<Dest>FromRIB. The following example uses the TAFR ewASNInToWHFromRIB.

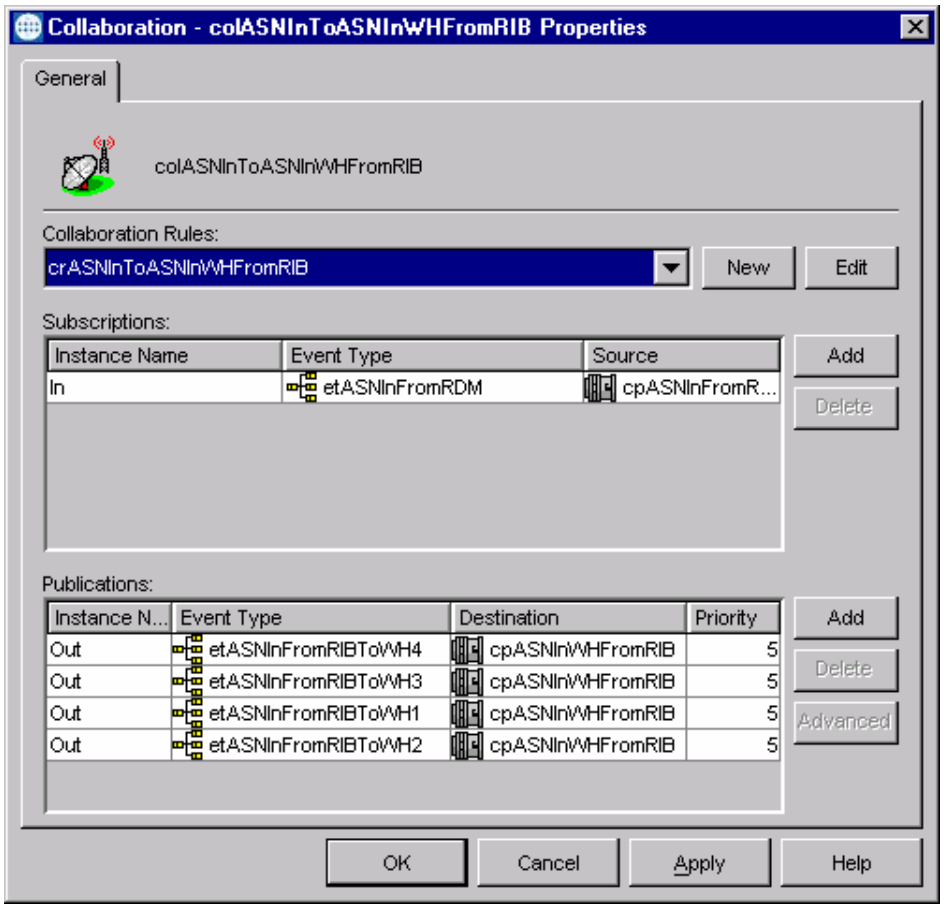


e*Gate Enterprise Manager with TAFR e*Way selected

2 Select an action:

- Double-click on the TAFR's collaboration.
- Select the TAFR's collaboration and click on the Properties icon in the toolbar.

3 The Collaboration Properties window is displayed.



Collaboration Properties window

To add the new event as valid for publication:

4 In the Publications section, click **Add**.

5 Duplicate the connection point specified as the destination.

6 Select the new event type to be published.

In the example, you would use the event type `etASNInFromRIBToWH5`.

Note: The “Destination” (in this case ‘WH5’) must *also* be found in the rib.properties file as a valid translation value for a specific facility ID code.

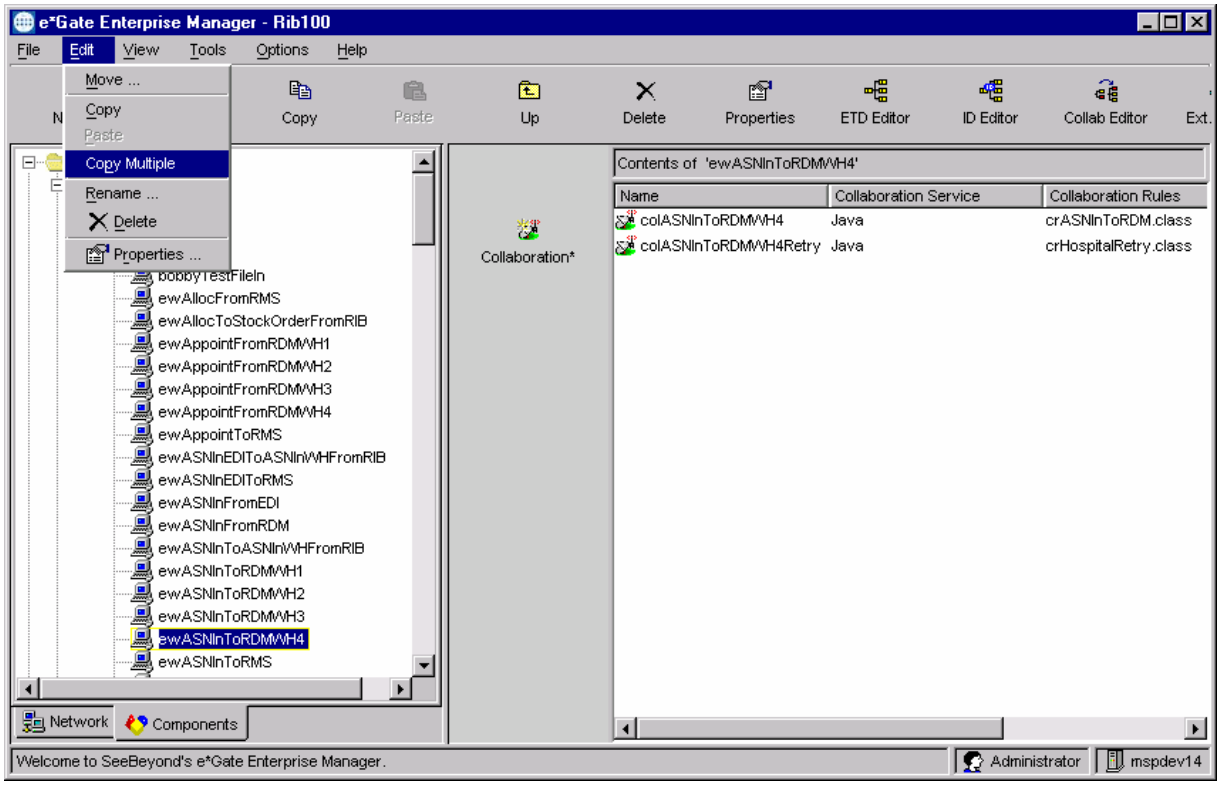
7 When the new event publication has been specified, click **OK** to save the information and update the e*Gate Registry with the new information.

Step 4: Create the destination messaging components

The last step is to create the subscribing RIB adapter. One way to do this is:

1 Select an e*Way to duplicate.

2 Select Edit > Copy multiple.

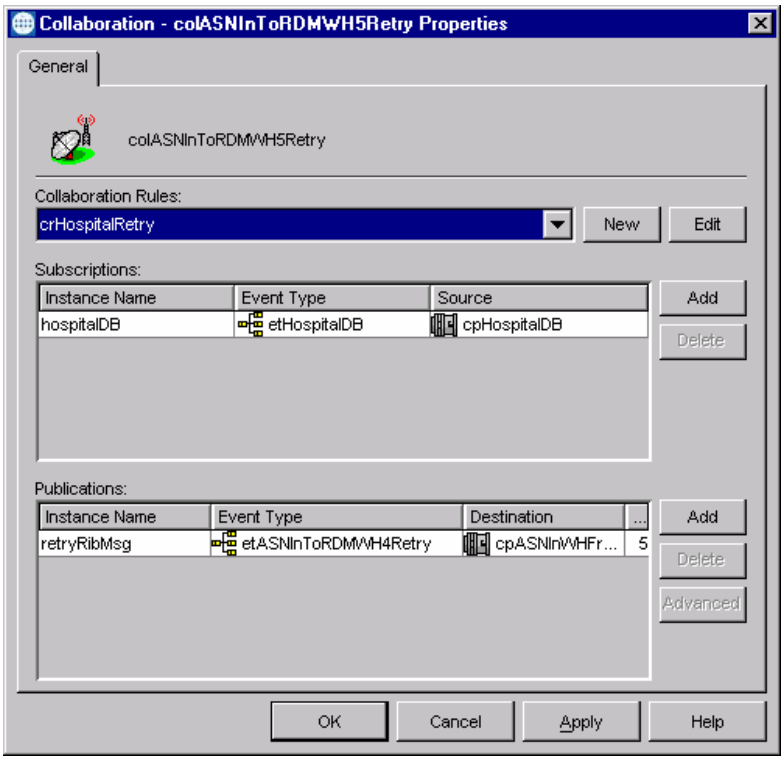


Copy Multiple edit option

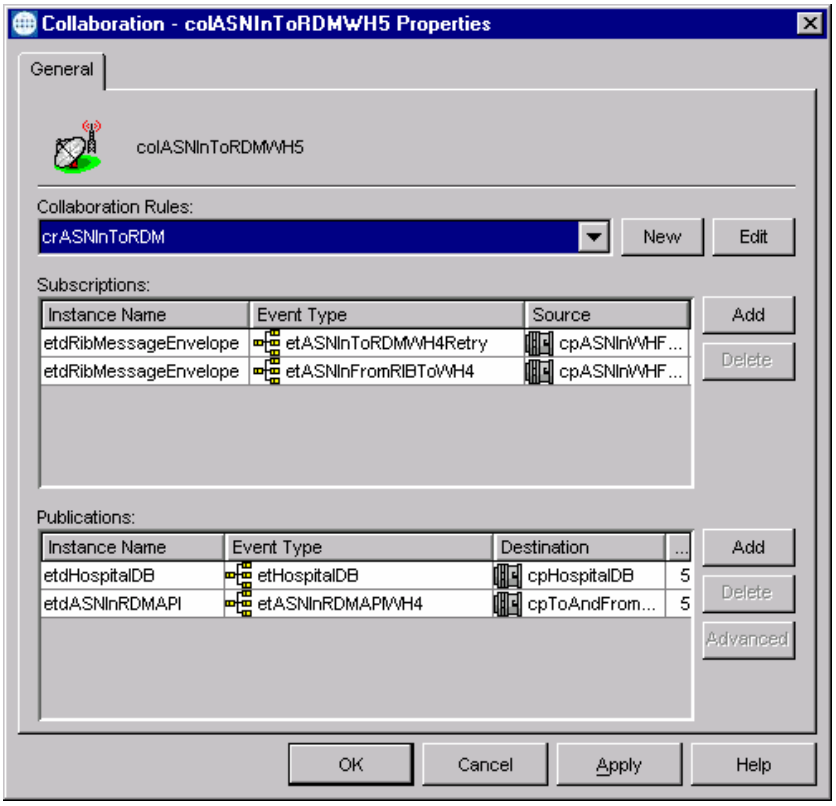
3 Rename the duplicate e*Ways to match the RIB's naming convention: For example, duplicating `ewASNInToRDMWH4` will result in `ewASNInToRDMWH4_0`. The RIB Naming convention renames the new e*Way to `ewASNInToRDMWH5`.

4 Rename the collaborations used to match the RIB naming convention.

5 Edit each collaboration in the Properties window.



Collaboration Properties window for a Subscribing Application Retry collaboration.



Collaboration Properties window for the subscribing collaboration for a Subscribing Application adapter

Note: This collaboration updates the application database.

The following must be changed on *both* collaborations:

6 Change the Event Type Names to match the new Event Types defined.

If you do not do this, the adapter will only receive messages that go to a different destination. In the example above, we created a warehouse #5. All references to the Event Type etASNInToRDMWH4Retry must be changed to etASNInToRDMWH5Retry and references to etASNInFromRIBToWH4 changed to etASNInFromRIBToWH5.

7 If the Error Hospital used is specific to the subscribing application, then make the connection point specific to the error hospital used.

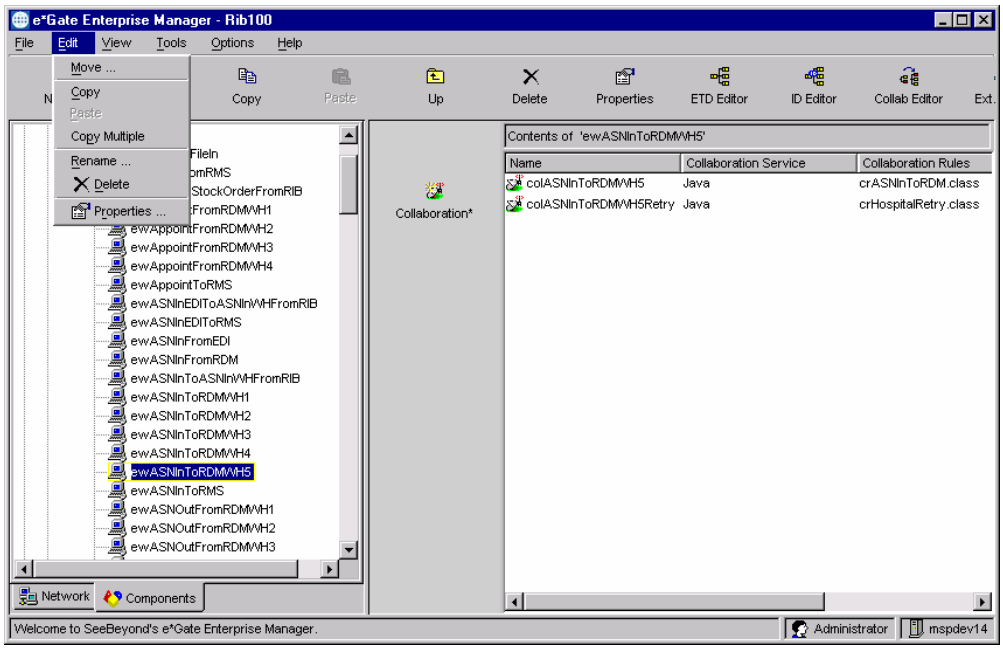
This connection point is associated with the etHospitalDB Event Type processing.

8 If the subscribing application is to be hosted by a different participating host, move the new e*Way:

a Select the adapter that you want to move.

b Select Edit > Move. Another window is displayed that allows the e*Way to be executed on a new computer.

The new computer must have an associated "Participating Host" created within an e*Gate Schema. See the SeeBeyond e*Gate Integrator User's Guide for more details. In addition, a running `stccb` daemon must be active on the computer before any other component can be run on the new participating host.



Edit drop-down menu

Note: You must select the e*Way to be moved before you select Edit > Move…

Chapter 5 – Message error handling

An error occurring while a subscriber processes a message poses a problem for an EAI system. If the error is one such as a broken database connection, the message simply needs to be retried once the connection is re-established. In these types of errors, one would like the message to remain on the EAI queue until it can be successfully processed.

Another type of error arises when messages have dependencies on seed data found in the subscribing database. For example, only the SKU number may reference a SKU referenced in a Purchase Order. If the subscribing database does not contain this SKU, an error will occur. This category of errors, referred to as *Message Content Errors*, cannot be resolved only through re-submitting the same message. Instead, the SKU must be added before the message can be successfully re-processed.

For the subscribing PO adapter, however, it may make sense to re-process the message a set number of times anyways. The message that creates a new SKU may be published by a different adapter than the one creating the Purchase Order. Because of possible performance bottlenecks or operational difficulties, the Purchase Order may arrive at the subscribing application adapter for POs before it arrives at the subscribing application adapter for SKUs. Therefore, simply re-trying the message gives the application an opportunity to successfully process the PO.

Once a Message Content Error occurs, it is desirable that the failing message does not affect the processing of other messages on the queue which refer to a different business entity. Messages not yet processed could contain acceptable data and it makes no sense to delay their processing. In order to get at these messages, the problem message must first be removed from the queue and, once removed, needs to be stored externally from the integration bus.

This storage mechanism is called the “Error Hospital”. Error Hospitals are associated with subscriber adapters. Subscribing adapters may share the same Error Hospital tables, or may have a set of tables reserved only for their specific use. Messages are re-submitted to the EAI queue by the subscriber and the resubmitted message will only be re-processed by the subscriber that resubmitted it.

If a message contains invalid data and there are three subscribers for this message family, then each subscriber will store a copy of the message in an Error Hospital and re-publish the message to the queue. We use message selectors on each subscriber that help filter the messages retried to the correct subscriber, as the retry e*Way publishes the message with the name of the collaboration on a property for retrying.

Each subscriber stores its own copy of the failing message because a different subscriber may have processed the message successfully. When the message is re-tried, those successful subscribers should not re-process the message.

Another complication with Message Content Errors is that subsequent messages within the same message family may have dependencies on the problem message. For example, a “Create New PO” message may be followed by an “Update PO” message for the same PO number. If the “create” cannot be processed, then the subscriber will error processing the “update”. Thus, before any message is processed, a check is performed to see if the Error Hospital already contains messages for the same business entity (in this case, the same Purchase Order). If so, then the follow-on message is immediately inserted into the error hospital, without allowing the application to process it at that time. The adapter should re-publish the follow-on message only after the first one has been successfully consumed by the application.

The retry logic for a publishing error hospital is much different than a subscribing error hospital, as the publishing error hospital directly calls the oracle package that failed for a message with the correct context data that is in the hospital, the retry call then attempts to retry the message. If it is successfully published, the message is removed from the error hospital database.

Once a message in the error hospital that had a dependency error message in the hospital is completed, then the publishing retry e*Way publishes this dependent message straight from the error hospital, as it was already successfully published from the database but couldn't be put on the rib because of sequencing.

Error Hospital components

Error Hospitals consist of a collection of Java classes, a set of database tables, a Connection Point providing access to these tables, and a "retry" collaboration. The Java classes contain the Error Hospital logic and include database access logic. The Connection Point must be configured for each subscriber and connect to the database housing the Error Hospital. The same Error Hospital Connection Point must be used between the "Normal" subscribing collaboration and the "retry" collaboration.

There is also a command line and a Graphical User Interface (GUI) tool for monitoring and manipulating messages found in the Error Hospital.

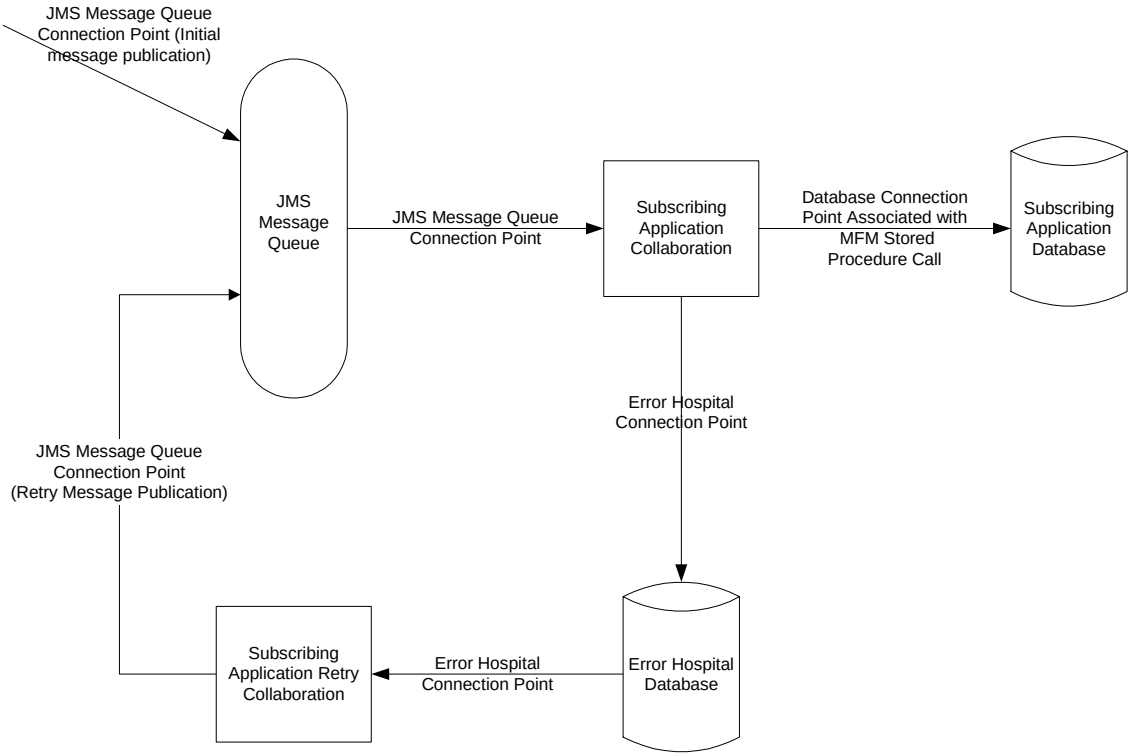


Figure 6-1 Connection Points used at a subscriber.

The following tables are used to store message information within 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 and information used to track a message's retry status.
- 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.

More information about the Error Hospital design may be found in the Retek Integration Bus Technical Architecture Guide.

Error Hospital configuration parameters and properties

All configuration parameters for an Error Hospital that control its logic are found in a properties file. This file must be part of the Java CLASSPATH used when the adapter is running. In the supplied Retek Messaging Schema, this properties file is named `rib.properties`.

The properties file, along with the name of the Java Archive (JAR) file containing Error Hospital classes and subscribing adapter helper classes, is specified in the adapters configuration file.

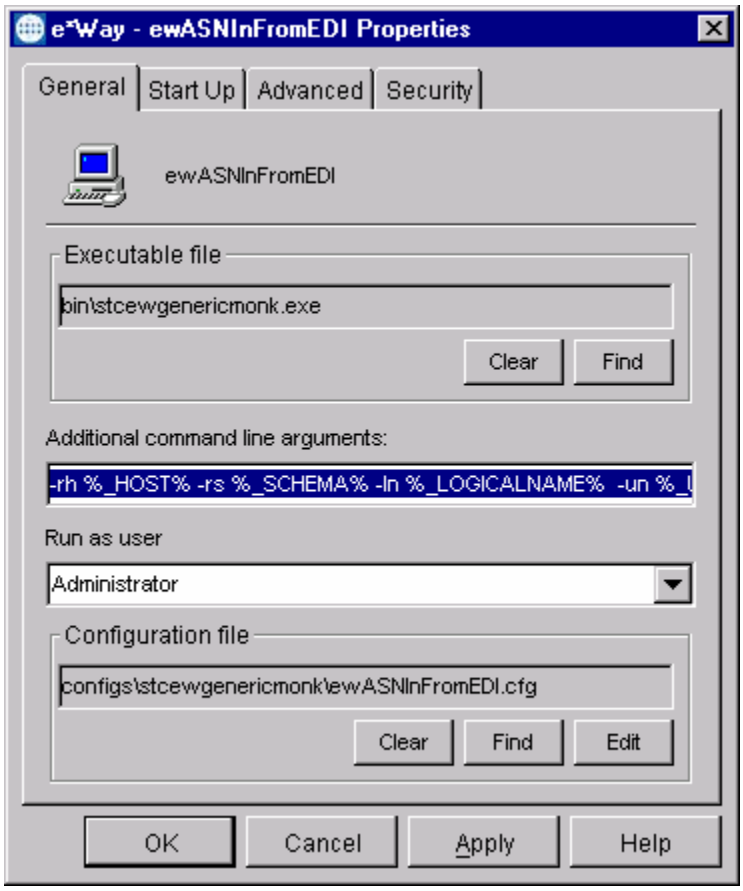
To access the adapter configuration:

1 Open the SeeBeyond Enterprise Manager.

2 Select an option:

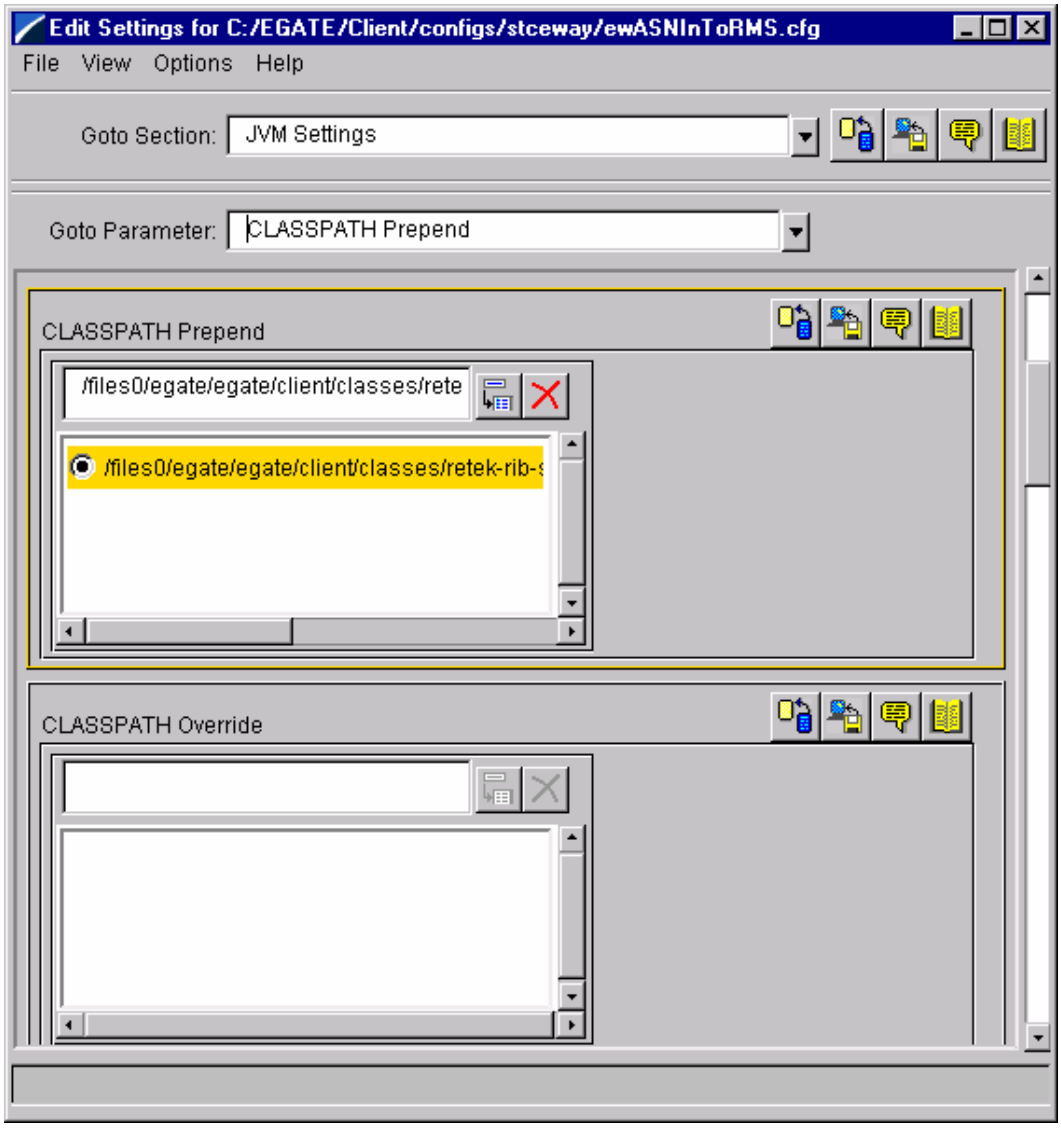
- Right click on the appropriate subscribing e*Way and select Properties.
- Select the appropriate subscribing e*Way and then click the Properties toolbar button.

The e*Way Properties dialog box is displayed.



e*Way Properties dialog box

3 In the Configuration file area, click **Edit**. The configuration file edit window is displayed. The CLASSPATH specification is found in the JVM Settings section under the CLASSPATH Prepend parameter.



Configuration file edit window

Note: If any parameter found in the configuration file is changed, an additional step is needed before the running system actually uses the new configuration: the configuration must be "Promoted to run-time". This may be done in the configuration file "File" drop-down menu or in the Enterprise Manager "File" drop-down menu. Simply changing a configuration does automatically update the SeeBeyond Registry with the new value.

The RIB properties file contains a number of parameters controlling the Error Hospital retry logic. Each parameter is on a line by itself and each line has the following form:

```
hospital.attempt.<param_name> = <param_value>
```

where `<param_name>` is the name of the parameter and `<param_value>` is the value. The table below lists the hospital parameters and their default values if not found in the RIB properties file:

Parameter Name	Default Value	Description
hospital.attempt.max	4	Maximum number of attempts the Error Hospital will make for the message, including the initial attempt. Once a message has been attempted this many times, a User Defined Alert is raised for this message. These alerts will seen on the e*Gate Monitor application.
hospital.attempt.delay	2	Base number of seconds between retries.
hospital.attempt.delayIncrement	8	Number of seconds to add to the base delay per each retry. For example, using the default value, the time between the third and fourth retry is: 2 + 8 + 8 + 8 = 26 seconds.

If different subscribers need different Error Hospital configurations, then each subscriber should use a different properties file with the values needed by that subscriber.

Note: Although the directory containing the RIB properties file may change, it must always be named `rib.properties`.

Error Hospital activities

This section details activities one may perform to messages in the Error Hospital from either the Hospital Administration GUI or the Hospital command line utility. This Java application lets you:

- Query the hospital database to determine the message(s) that exist
- View or save a message's contents
- Replace the message's contents
- Increase the number of processing attempts for this message for this subscriber by one
- Delete the message
- Stop the message from further processing attempts

The Hospital GUI and command line utility are Java classes that are executed or wrapped by a set of shell scripts (Unix) or BAT files (Windows/NT). This Java class requires the presence of a properties file, `hospital-admin.properties`, in the user's home directory.

These scripts also source the file, `hospital-admin.env`, to initialize the CLASSPATH used by the command line utility class.

Hospital command line utility set up

The hospital-admin.properties file and the hospital-admin.env file must be manually set up before the GUI or command line utility can be used. This is detailed in the next section.

Setting up hospital-admin.properties

The following properties must be set in the file `hospital-admin.properties`. By default, the user's home directory is checked for this file. However, the name and location for this file may be specified at run time.

Parameter Name	Description
hospital.gui.prop.dbUser	Database User ID the utility will use to log into the hospital database.
hospitial.gui.prop.dbPwd	Password associated with the **dbUser** parameter.
hospital.gui.prop.dbUrl	URL of the JDBC driver that will host the database session. This URL is typically of the form: jdbc:oracle:thin:@<**hostname**>:1521:<**SID**> where <**hostname**> is the name of the host containing the Oracle listener and <**SID**> is the Oracle System ID of the database.
hospital.gui.prop.dbDriverClass	Name of the Oracle JDBC driver class. Typically, this is **oracle.jdbc.driver.OracleDriver** As of this writing, this driver is found in the file **client12.zip** available from Oracle.

Because this file contains database login parameters, access to it should be limited. On Unix systems, set the file privileges mode of `hospital-admin.properties` to 0400.

All entries must be in the form <ParameterName> = <Value>. Comments begin with a hash ('#') and continue to the end of a line. Lines containing white space are ignored. An example of the `hospital-admin.properties` follows:

```
hospital.gui.prop.dbUser=retek_user
hospital.gui.prop.dbPwd=retek_password
hospital.gui.prop.dbUrl=jdbc:oracle:thin:@HSP_DB_HOST:1521:hsp_SID
hospital.gui.prop.dbDriverClass=oracle.jdbc.driver.OracleDriver
```

Setting up hospital-admin.env

The hospital-admin.env file contains the CLASSPATH and other environment entries that the hospital command line utility uses. Each wrapping [?] script sources this file before executing the utility class. The `hospital-admin.env` file must exist somewhere in the user's execution path.

The hospital-admin.env file should contain the following information:

- The correct CLASSPATH environment variable. An example of a CLASSPATH is:

  ```
  CLASSPATH=/files0/egate/egate/client/classes/retek-rib-
  support.jar:/files0/egate/egate/client/ThirdParty/oracle/classes/cla
  sses12.zip:/files0/egate/egate/client/etd/etdRibMessageEnvelope.jar:
  /files0/egate/egate/client/classes/stcjcs.jar
  ```

  The example above assumes that the <EHOME> directory is /files0/egate/egate.

- Any modifications to the PATH environment variable to execute the Java command.

Error Hospital admin command line scripts

All Error Hospital administration is done via the Java class:

```
com.retek.rib.collab.HospitalAdminCmdLine
```

However, a set of scripts has been created for ease of use. These scripts wrapper the HospitalAdminCmdLine class and invoke the java interpreter to execute it. Each script will also echo the specific command used.

Each script has a Unix Bourne shell version and a Windows 2000/NT version. Each operating system specific version accepts the same parameters. The following scripts have been implemented:

Command	Parameters	Description
querymsg	-l <location> -f <family> -t <type> -i <id> -q <inQueue> -r <willRetry> -p <propertiesFile>	Queries the database and displays a list of message numbers that meet the required criteria. Any combination of these parameters can be used. The SQL select will use the input parameters in a LIKE context so wildcards are allowed (%). For example, if "-i 123%" were passed in, all messages with message_num starting with 123 would be selected. • -l <location> lists only those message numbers from the specified location. Locations are of the form <eway name>.<collaborationName> • -f <family> lists only those message numbers belonging to the specified message families • -g <type> lists only those message numbers that belong to messages

Command	Parameters	Description
		of the specified type • -i <id> lists only those message numbers that apply to the specified ID. These identify a specific business object, such as a Purchase Order or ASN. • -q <inQueue> lists only those message numbers that are believed to be enqueued in the integration bus at the current time. A value of 0 or "false" implies the message only exists in the Error Hospital, a value of 1 or "true" implies that the message is thought to have been published for another attempt to process it. • -r <retry> lists only those messages according to their retry status. The <retry> specification of 0 or "false" lists those not eligible for retry and marked ready for delete; a value of 1 or "true" lists those eligible for retry and not ready to be deleted. All parameters are optional. Multiple parameters produce the intersection of their independent results. (For example, –f Family and –l Location lists all messages in family "Family" belonging to location "Location".)
deletemsg	-m <messageID> -p <propertiesFile>	Marks the message ready for deletion. The message will be deleted when the retry collaboration next awakens. The –m switch is mandatory and must contain the message number of the message to delete.
readmsg	-m <messageID> -F <outputFileName> -p <propertiesFile>	Retrieves the payload contents for message <messageID> and writes it out to the file <outputFileName>. The –m switch is mandatory and must contain the message number of the message to read.
updatemsg	-m <messageID> -f <inputFileName> -p <propertiesFile>	Replaces the message payload for the given message with the contents of the file. No validation of the file contents is

Command	Parameters	Description
		performed until the subscribing adapter processes the data. The –m switch is mandatory and must contain the message number of the message to update.
stopmsg	-m <messageID> -p <propertiesFile>	Stops further attempts to retry the message. The –m switch is mandatory and must contain the message number of the message to stop retrying.
retrymsg	-m <messageID> -p <propertiesFile>	Flags the message so one additional attempt is made to process the message. The –m switch is mandatory and must contain the message number of the message to retry.

Hospital Administration command line examples

Before using any of the commands below, remember to verify that the `hospital-admin.properties` file exists in your home directory and contains the correct database login information. The name and location of this file may be overridden via the **–p** command line switch.

Listing all messages in an Error Hospital:

```
> querymsg

[USAGE] querymsg [-p properties file] [-l location] [-f family] [-t
type] [-i id] [-q inQueue] [-r willRetry]

java HospitalAdminCmdLine -a query

Getting properties from: /files0/egate/hospital-admin.properties

Number of messages selected: 159

Message numbers:        2947    2933    2934    2935    2936    2940
2849    2850    2851    2852    2853    2854    2856    2857    2858
2859    2923    2924    2925    2926    2927    2928    2929    2930
2931    2932

SUCCESS
```

Listing all messages in an Error Hospital from a specific e*Way:

The example below lists all message numbers that belong to the ewASNOutToRCOM e*Way:

```
> querymsg -l ewASNOutToRCOM%

[USAGE] querymsg [-p properties file] [-l location] [-f family] [-t
type] [-i id] [-q inQueue] [-r willRetry]

java HospitalAdminCmdLine -a query -l ewASNOutToRCOM%

Getting properties from: /files0/egate/hospital-admin.properties

Number of messages selected: 15

Message numbers:        2854    2913    2804    2805    2809    2811
2813    2769    2794    2795    3113    3115    3117    3119    3124

SUCCESS
```

Listing all messages in an Error Hospital that belong to a specific message family:

The example below lists all message numbers that belong to the “asnout” message family:

```
> querymsg -f asnout

[USAGE] querymsg [-p properties file] [-l location] [-f family] [-t
type] [-i id] [-q inQueue] [-r willRetry]

java HospitalAdminCmdLine -a query -f asnout

Getting properties from: /files0/egate/hospital-admin.properties

Number of messages selected: 23

Message numbers: 2854   2913    2804    2805    2808    2809   2810
2811    2812     2813   3019    3045    3012    2769    2794   2795
3205    3226    3113    3115    3117    3119    3124

SUCCESS
```

Reading the message payload XML into a file:

Message contents can be read into a file using the `readmsg` script. Note that the XML is written as it appears in the original message and this means it contains no new-line or carriage return characters.

```
> readmsg -m 2947 -F /tmp/message_2947.XML

java HospitalAdminCmdLine -a read -m 2947 -F /tmp/message_2947.XML

Getting properties from: /files0/egate/hospital-admin.properties

read Message: 2947

SUCCESS

> cat /tmp/message_2947.XML

<?XML version="1.0" encoding="UTF-8"?><!DOCTYPE POReceiptDesc SYSTEM
"http://mspdev09:8109/dtdtst/POReceiptDesc.dtd"><POReceiptDesc><dc_d
est_id>1</dc_dest_id><appt_nbr>500000301</appt_nbr><po_nbr>10610</po
_nbr><document_type>P</document_type><item_id>100614114</item_id><un
it_qty>8</unit_qty><receipt_xactn_type>R</receipt_xactn_type><receip
t_date><year>2002</year><month>03</month><day>08</day><hour>16</hour
><minute>47</minute><second>11</second></receipt_date><receipt_nbr>5
00000291</receipt_nbr><asn_nbr>ASN-IT-RECEIPT-
19</asn_nbr><dest_id>1000000014</dest_id><container_id>ASN-IT-REC-
19-
CID001</container_id><distro_nbr>1000001911</distro_nbr><distro_doc_
type>A</distro_doc_type><to_disposition>WIP</to_disposition><from_di
sposition></from_disposition><to_wip>MXDSKU</to_wip><from_wip></from
_wip><to_trouble></to_trouble><from_trouble></from_trouble><user_id>
ZZRUDEJ</user_id></POReceiptDesc>
```

Updating the message payload from a file:

Message contents can be updated from a file using the `updatemsg` script. The editor used to manipulate this data is external to this application.

```
> updatemsg -m 2947 -F /tmp/message_2947.XML

java HospitalAdminCmdLine -a update -m 2947 -F /tmp/message_2947.XML

Getting properties from: /files0/egate/hospital-admin.properties
```

```
update Message: 2947
SUCCESS
```

Marking a message ready for deletion:

The deletion of messages stored in the Error Hospital is performed by the retry collaboration. One may mark a message ready to be deleted by this software using the `deletemsg` script. The example below marks message number 2155 ready to be deleted:

```
> deletemsg -m 2155
java HospitalAdminCmdLine -a delete -m 2155
Getting properties from: /files0/egate/hospital-admin.properties
delete Message: 2155
SUCCESS
```

Manually querying message information from Error Hospital

Although the Hospital Admin command line utility allows one to view information about the messages contained in the hospital, one may wish to select IDs from the Error Hospital database using some other unique criteria.

Most message information is stored in the `rib_message` table.

To count the number of messages in the Error Hospital for a specific adapter:

```
select count(*) from rib_message where location = '<ADAPTER_NAME>';
```

To display the Error Hospital message numbers for messages in the Error Hospital for a specific adapter:

```
select message_num from rib_message where location =
'<ADAPTER_NAME>';
```

To display the failure history of a specific message

```
select * from rib_message_failure where message_num = <MESSAGE_NUM>;
```

To display the message numbers for a particular message type

```
select count(*) from rib_message where location = '<ADAPTER_NAME>';
```

Columns in the RIB_MESSAGE table

Column Name	Description
message_num	Error Hospital message ID
Location	Name of adapter (e*Way name + '.' + collaboration name) encountering an error processing the message
family	family of message
type	type of message
ID	ID of business entity that this message is associated with
ribmessageID	RIB ID of message. Contains RIB version, publishing e*Way name, collaboration name, e*Way start time and a unique sequence ID.
publish_time	Date/Time message published
in_queue	Flag set when message is re-published by the retry collaboration. A value of 1 indicates the message resides on the JMS queue and has not yet been processed by the subscriber collaboration. A value of 0 indicates the message only resides in the Error Hospital
message_data	CLOB containing the message data
attempt_count	The number of times this message has been sent (unsuccessfully) to the subscriber, including the initial attempt

Column Name	Description
max_attempts	The number of attempts the hospital will make before stopping retries and alerting an administrator
next_attempt_time	The time of the next retry attempt, or null if the message should be attempted as soon as possible.
delete_pending	Set to 0 to indicate message is to be kept in the Error Hospital. Set to 1 to prompt the retry collaboration to delete the message from the Error Hospital.

Error Hospital log entries

The Error Hospital software contains trace statements for monitoring its execution. These statements will be logged to the e*Way RIB log files. More verbose logging of hospital operations is available if the e*Way's verbose log settings have been set to Y in the rib.properties file.

The log filename will be (rib_<EWAY_NAME>.log) and it will be written to the default log directory as specified in the rib.properties file.

Create additional Error Hospitals

An Error Hospital is checked each time a subscribing application adapter processes a message. Because of this, location of the database with the Error Hospital tables is critical. The Error Hospital may be located within its own database or be part of the application's database.

By default, only a single Error Hospital is used in the RIB Messaging schema. The instructions for installing a new Error Hospital are found in the RIB Installation Guide. This installation consists of creating a set of new database tables and a sequence object.

Once the new Error Hospital has been created, create a new Oracle Connection Point to reference it. Then update the collaborations used by the subscribing application adapters to use the new Connection Point

Chapter 6 – J2EE Platforms

RIB startup and shutdown

This section details considerations for bringing up and shutting down the RIB Enterprise Java Beans and Message-Driven Beans when deployed on a J2EE Application Server, such as WebSphere.

Starting the RIB components

All RIB EJB components should be automatically started when the application server is brought up. One prerequisite is for the SeeBeyond JMS IQ Manager to be running before starting the Application Server.

The SeeBeyond JMS server, however, requires a SeeBeyond instance. If the JMS is not available, then follow the instructions for configuring the SeeBeyond RIB Components.

Shutting Down RIB Components

With the exception of the SeeBeyond JMS Server, all RIB components should cease to function once the J2EE Application Server is brought down or the Application is stopped.

Preventative maintenance tasks

Log files are the primary tools used to determine activity. These files must be maintained as they could continue to grow to unmanageable sizes.

Log Files

WebSphere log files

WebSphere's log files are managed from its Administration Console. You can configure the maximum size of the files, the number of histories to keep, etc. Refer to WebSphere for the details of these configurations.

RIB/Timings log files

The RIB/Timings logs are not managed and must be maintained. These files are configured using Log4J. See the RIB Installation Guide, Chapter 6 for more information on configuring these log files using Log4J.

RIB component configuration

This section will detail configuration information used in a WebSphere J2EE environment. See Chapter 6 of the RIB Installation Guide for more details on configuring this environment.

Configuration files

rib.properties

RIB configuration file. See the RIB Installation Guide, Chapter 6 for details on the rib.properties entries required for a J2EE environment.

log4j.properties

Configures logging. See the RIB Installation Guide, Chapter 6 for details on the log4j.properties entries required for a J2EE environment.

.bindings

In the …/WebSphere/sbynjndi directory you will find a file named "**.bindings**". This hidden file contains the serialized java JMS Objects that the Generic JMS Provider uses. It is created as part of the RIBfor*APP* installation, detailed in Chapter 6 of the RIB Installation guide.

Note: For RIBforMDM the directory is …/WebSphere/AppServer/rib/sbynjndi

hibernate.cfg.xml

Hibernate configuration file. See the RIB Installation Guide, Chapter 6 for details on the hibernate.cfg.xml entries required for a J2EE environment.

Generic JMS Provider

The Generic JMS Provider is fully configured as part of the RIBfor<*APP*> installation. From the WebSphere Admin Console, click Resources -> Generic JMS Providers. You will see "**SeeBeyond JMS Provider**" as the available resource. The JMS Connection Factory as well as all the JMS Destinations are defined here.

Message Listener Ports

The Message Listener Ports are also fully configured as part of the RIBfor<*APP*> installation. From the WebSphere Admin Console, click Servers -> Application Servers -> server1 -> Message Listener Service -> Listener Ports. You will see all of the WebSphere Listener Ports defined here.

Data Source

The Oracle DataSources are fully configured as part of the RIBfor<*APP*> installation. From the WebSphere Admin Console, click Resources -> JDBC Providers. You will see "**Oracle JDBC Thin Driver (XA)**" as the available resource. All of the <*APP*> DataSources are defined here. The "**Oracle Rib Datasource**" is the DataSource that the RIB utilizes.

Error Hospital Retry

Finally, the Error Hospital Retry EJB may be deployed as part of the RIBfor<*APP*> installation. This can be configured and administered through the web browser.

http://<server>:<http port>/ribhospitalretry/ErrorHospitalRetryServlet

Chapter 7 – ISO Platform

ISO application server

The ISO application was patterned after the specifications for the J2EE application server, though it was developed as the specifications evolved, long before they were complete. For that reason, it is not J2EE compliant. However, though the terminology may be different, some of the same concepts apply. The application server has containers that hold server components, which are EJBs in J2EE. ISO has messaging components, while J2EE has message-driven beans. ISO has configuration files, while J2EE has deployment descriptors.

The ISO application server was designed with flexibility of deployment in mind. There are none of the "per PC" licensing requirements that traditional application servers, such as WebSphere, have. Also, it doesn't require a heavy-duty server to run it. This is important for a large retailer who has many individual store locations, each of which requires an application server.

The ISO application server can use a SeeBeyond JMS queue manager as its JMS messaging service. In fact, this is the JMS implementation that the RIB uses for integration between ISO, and other Retek modules such as RMS and RDM. The existing Retek publishers and subscribers are still SeeBeyond e*Ways, however, the new ISO components are ISO messaging components for its subscribers, and publishing utilities.

For more information on the ISO application server, see the documentation supplied with the SIM/ISO application.

ISO-specific Components

If you have purchased the SIM/ISO module, in addition to e*Ways, you will have ISO platform messaging components that can be monitored using the Mission Control application, which is part of the ISO application. Within Mission Control, the highest level entity that can be monitored is the container. By default, ISO RIB components come in their own container, separate from the components that are part of the ISO application. The containers can be monitored to determine whether they are currently up or down, and how long they have been running. Other miscellaneous vital statistics can also be viewed from Mission Control.

Within each container in Mission Control, individual components can be monitored to determine whether they are currently up or down, how long they have been running, their transaction counts, and any error messages can be viewed as well.

RIB startup and shutdown

Starting the Rib components for the ISO application is as easy as starting the ISO application server. No additional steps are necessary, as long as the configuration files have been installed correctly in the Rib install process. See "Chapter 14 – RIB component configuration: ISO Platform" for details regarding the configuration files.

Preventative maintenance tasks

This chapter lists some common tasks that a system administrator may want to script and perform on a regular basis, or may not need to script or perform on a regular basis.

Log files

Each of the subscribing Rib messaging components has a log file associated with it. Each publisher, although not a server component, is associated with a particular message family, and has its own log file as well. The names of these log files are set in the configuration file for the subscriber or publisher. Also contained in the configuration files are some Log4j logging properties that can be used to configure the maintenance of these log files. For more information on Log4j, see the documentation at the following Internet URL:

http://jakarta.apache.org/log4j/docs/documentation.html

There are four entries in the publisher and subscriber configuration files that deal with log file maintenance. The names of these properties are:

- LOGGING_LOG4J_LEVEL
- LOGGING_LOG4J_MAX_FILE_SIZE
- LOGGING_LOG4J_MAX_BACKUP_INDEX
- LOGGING_LOG4J_PATTERN_FORMAT

The first entry has to do with the level of detail that will be output to the log file. The log file will grow most quickly if the level is set to "DEBUG". To keep the log files smaller, you may want to set the level to a different value. The default is "DEBUG".

The second entry has to do with the maximum size to which a log file is allowed to grow. Once the file reaches this size, if the value for the third property, LOGGING_LOG4J_MAX_BACKUP_INDEX, is positive, then files {File.1, ..., File.MaxBackupIndex -1} are renamed to {File.2, ..., File.MaxBackupIndex}. Moreover, File is renamed File.1 and closed. A new File is created to receive further log output. If MaxBackupIndex is equal to zero, then the File is truncated with no backup files created. This allows an administrator to maintain the log files with no scripting required.

The default value (the value that is in the configuration file to start with) of the second property is, "1024KB", or one megabyte. The default value for the third property is "1".

The last property, "LOGGING_LOG4J_PATTERN_FORMAT", controls the format of the output data. For more information on this setting, see the documentation at the following Internet URL:

http://jakarta.apache.org/log4j/docs/documentation.html

RIB component configuration

XML files

RibContainer.xml

The key XMLconfiguration file for the ISO application server is **RIBContainer.xml.** This file will be found in one of the following directories:

Unix:
```
<install_dir>/chelsea/serverUnix/retek/sim/files/prod/tuning
```

Windows:
```
<install_dir>\chelsea\serverWdws\retek\sim\files\prod\tuning
```

This configuration file **must** be present in this directory in order for the RIB components to be deployed. There needs to be an entry in RIBContainer.xml file for each of the messaging components (subscribers).

Some of the other key entries in this file are:

For the container as a whole:

- **containerName –** This entry controls the naming of the container's log files, and the name displayed in the Mission Control application for the RIB's container. It is "RIBContainer" by default.
- **defaultInstanceCount –** This entry controls how many instances of the container are started at startup. It is set to "1" by default.
- **MinutesPauseVitals –** This entry controls the delay updates to the container's vitals in the Mission Control application. The default is "5".

For the individual components:

- **componentClassName –** This entry controls the class that the component consists of. This class *must* be a descendant of com.chelseasystems.cr.node.CMSComponent. The default is com.retek.rib.redsky.RibMessagingComponent. This entry should not normally need to be changed.
- **defaultMaxCount –** This entry controls the minimum number of instances of the component that will be allowed to exist. If the number of instances of the component ever dips below this number, a new instance of the component will be created.
- **defaultMinCount –** This entry controls the maximum number of instances of the component that will be allowed to exist. If the number of instances of the component ever goes above this number, an instance of the component will be destroyed.
- **name –** This entry controls the name of the component, as displayed within the Mission Control application.
- **propertyPairs –** This entry controls what name/value pairs, or properties, are passed to component upon startup of the component. Of all the standard name/value pairs available, one is mandatory. It is, "CONFIG_FILE", and its value should be the name of the configuration file for the component. No path information should be included with this value, as ISO will look for this file in the standard "config" directory. For the RIB components, this is the only entry that is necessary.

Retek Binding Mappings

Retek Binding Mapping XML Files detail the mapping of the XML data to/from the payload object. They exist mainly to prevent costly message validation.

ISO Configuration (*.cfg) files

Non-XML formatted configuration files for the RIB on the ISO application server platform are:

Subscriber messaging component configuration

Subscribing messaging component configuration files use the following naming convention: **<RibFamilyName>messagingcomponent.cfg**

An instance of this file should exist in the ISO "config" directory, for each RIB component deployed. Remember, the messaging components represent subscribers, and as such they are server components that are brought up when the application server starts up. The names of the configuration files for the standard RIB components include:

- asnoutmessagingcomponent.cfg
- diffsmessagingcomponent.cfg
- itemsmessagingcomponent.cfg
- ordermessagingcomponent.cfg
- seedmessagingcomponent.cfg
- storesmessagingcomponent.cfg
- vendormessagingcomponent.cfg
- whmessagingcomponent.cfg

Some of the key entries in these subscriber configuration files are:

- **TOPIC_NAME –** The value of this entry should be the topic name in SeeBeyond, to which the component subscribes.
- **DURABLE_SUBSCRIBER –** The value of this entry should be "true". All of the RIB's subscribing e*Ways in SeeBeyond are durable, and all of the ISO subscribers should be durable as well. For a definition of a durable subscriber, see the Sun JMS specification.
- **JMS_COMPONENT_TYPE –** The value of this entry should be "Subscriber". Remember, we are talking about the configuration files for ISO subscribing messaging components here.
- **MODULE_NAME –** The overall component name. For the RIB subscribers, this should be "RibMessagingComponent".
- **SUB_MODULE_NAME –** The RIB family name for the subscriber.
- **SINGLE_THREADED –** The valid values for this entry are "true" and "false". If this entry is set to "true", only a single thread will be used to call the processMessages(ArrayList) method. This method is the main method of the subscribing messaging component, and is responsible for consuming individual RIB messages. If the value for this entry is "false", multiple threads may call this method. The default is, "true".
- **MESSAGING_CONFIG –** The name of the JMS messaging configuration file. This path information should not be included in this file, as ISO will look in the standard "config" directory for this file. See **"JMS Messaging in General"**, below for more information on this file.
- **Logging - log4j –** There should be a section in the file for Log4j logging. The individual properties in this section are:
  - LOGGING_LOG4J_LEVEL
  - LOGGING_LOG4J_MAX_FILE_SIZE
  - LOGGING_LOG4J_MAX_BACKUP_INDEX
  - LOGGING_LOG4J_PATTERN_FORMAT

For a description of the individual entries, see the following Internet URL:
http://jakarta.apache.org/log4j/docs/documentation.html

Publisher messaging component configuration

Publishing messaging component configuration files use the following naming format: **<RibFamilyName>publisher.cfg.**

The publishers are utility classes, and although they require configuration files, they are not server components that are brought up during startup. Also, entries for these publishers are *not* required in the RIBContainer.xml configuration file. Names of the configuration files for the standard Rib publishers include:

- asnoutpublisher.cfg
- dsdreceiptpublisher.cfg
- invadjustpublisher.cfg
- receivingpublisher.cfg
- rtvpublisher.cfg

Some of the key entries in these publisher configuration files are:

- **TOPIC_NAME –** The value of this entry should be the topic name in SeeBeyond, to which the component publishes.
- **JMS_COMPONENT_TYPE –** The value of this entry should be "Publisher". Remember, we are talking about the configuration files for individual instances of the publisher utility class here.
- **MODULE_NAME –** The overall component name. For the Rib publishers, this should be "RibPublishingUtility".
- **SUB_MODULE_NAME –** The Rib family name for the publisher.
- **Logging - log4j –** There should be a section in the file for Log4j logging. The individual properties in this section are:
  - LOGGING_LOG4J_LEVEL
  - LOGGING_LOG4J_MAX_FILE_SIZE
  - LOGGING_LOG4J_MAX_BACKUP_INDEX
  - LOGGING_LOG4J_PATTERN_FORMAT

For a description of the individual entries, see the following Internet URL:
http://jakarta.apache.org/log4j/docs/documentation.html

- **JMS Messaging in General –** There is a configuration file for general JMS messaging for the Rib. Its name is ribmessaging.cfg, and it is located in the standard ISO "config" directory. There should be a property in each of the above configuration files, for both subscribers and publishers that refers to this file. The property name is, "MESSAGING_CONFIG" and its value should be, "ribmessaging.cfg". Some of the key entries in this file are:
  - **CLIENT_IMPL –** Should be "com.**retek**.rib.redsky.RibSeeBeyondJmsServices" for all Rib ISO subscribers and publishers.
  - **USE_SESSION_TRANSACTION –** The value of this entry should be "true". What this means is that the container session should control the entire **transaction**, rather than the individual database and JMS sessions within the overall transaction. What this amounts to a sort of two-phase commit, where the container session knows all of the individual database and JMS sessions involved, and the Rib messaging component tells the session to commit all involved sessions. This entry should always be "true".
  - **BROKER –** Should consist of the host name of the server on which SeeBeyond is running, plus ":", plus the port of the JMS queue manager. The port of the JMS queue manager can be found in the SeeBeyond e*Gate Enterprise Manager application. Navigate to the JMS queue manager, go to the "Properties" for it, and look under the "Advanced" tab.

Properties files

The property files for the Rib/ISO installation are:

- **binding.properties** – This is a "Retek Binding" subsystem file. It is located under the standard ISO "config" directory. Within the "config" directory, the pathname is, "com/retek/binding/rib/castor.properties". See the "Retek Binding Configuration files" section in "Chapter 4 – Configuration files" for properties files relating to the Retek Binding.
- **castor.properties –** This is a "Retek Binding" subsystem file. It is located in the standard ISO "config" directory. See the "Retek Binding Configuration files" section in "Chapter 4 – Configuration files" for properties files relating to the Retek Binding.
- **injector.properties –** This is a "Retek Binding" subsystem file. It is located in the standard ISO "config" directory. See the "Retek Binding Configuration files" section in "Chapter 4 – Configuration files" for properties files relating to the Retek Binding.
- **payload.properties** – This is a "Retek Binding" subsystem file. It is located under the standard ISO "config" directory. Within the "config" directory, the pathname is, "com/retek/binding/rib/castor.properties". See the "Retek Binding Configuration files" section in "Chapter 4 – Configuration files" for properties files relating to the Retek Binding.

- **publisher.properties –** This file is also used by the Retek Binding subsystem (see the "Retek Binding Configuration files" section in "Chapter 4 – Configuration files" for entries relating to the Retek Binding). Some additional entries are included in this file for the Rib publishers. The property names consist of the Rib message family name, plus ".", plus the Rib message type name. An example would be, "ASNOUT.ASNOUTCRE". The value for each of these properties would be the name of the configuration file for each of the publishers. The path information should not be included, as ISO will look for these configuration files in the standard ISO "config" directory. The properties and there values should be:
- ASNOUT.ASNOUTCRE=asnoutpublisher.cfg
- DSDRECEIPT.DSDRECEIPTCRE=dsdreceiptpublisher.cfg
- INVADJUST.INVADJUSTCRE=invadjustpublisher.cfg
- RECEIVING.RECEIPTCRE=receivingpublisher.cfg
- RECEIVING.RECEIPTMOD=receivingpublisher.cfg
- RTV.RTVCRE=rtvpublisher.cfg

- rib.properties – See the description under "ISO Platform Specific entries", under the "RIB Properties File" section of "Chapter 4 – Configuration files".

Chapter 8 – RIB Administration Tool

Overview

The RIB Administration Tool contains three administrative GUI Applets: a Hospital Administration GUI Applet, a RIB Properties Editor GUI Applet and a Message Statistics GUI Applet. This web application is contained in the gui.war file under the Rib_Hospital_Gui directory.

The application is for administration of a RIB installation on the same computer as the Application Server hosting the RIB Administration Tool.

Installation and configuration

The RIB Administration Tool requires an existing application server, such as Apache Tomcat, installed and running on the same host as the running RIB installation.

The RIB comes equipped with the necessary 'war' file, named 'gui.war', for the installation. This file is found <install_dir>/RIB103/Rib_Hospital_Gui/build directory.

To install and configure the RIB Administration Tool using the Tomcat application server:

1 Install the war file on an application server using the gui.war file.

2 Edit the gui.properties configuration file.

```
##############################################
# GUI Project Variables
GUI.ProjectHost=
GUI.ProjectPort=
GUI.ProjectName=
GUI.TimingsLogFile.Path=
GUI.TimingsLogFile.Name=
GUI.rib.properties.default.FilePath=
GUI.rib.properties.default.BackupFileExt=.bak
```

- GUI.ProjectHost and GUI.ProjectPort are values you can set for all the applets. These values override the applet's baseurl.getContext lookups to find the URL to the servlets. If for any reason this lookup does not find your correct host and port, or if you want to use a servlet residing on a different host or port, set these values in the properties file.
- GUI.ProjectName should be set in the properties file to contain the name of your project installation (installed application name) on the application server. The applets will use this name to build the URL to the servlets. The default installation name is "gui".

- GUI.TimingsLogFile.Path and the GUI.TimingsLogFile.Name should be set to contain the default path to the timings log file and the default name for the log file for the Message Statistics GUI Applet. When this applet is loaded, it will display a window where the user can enter the path to the log file and the parameters to pass into the RibTimings class to gather the statistics. The default path is displayed using these properties. If no value is entered, the log file path text field on this window will initially be blank.
- GUI.rib.properties.default.FilePath should be set to the default file path of the rib.properties file. This will be displayed in the RIB Properties Editor's connection window as the default File Name, which the user can modify before retrieving the file from the server.
- GUI.rib.properties.default.BackupFileExt should be set to contain the default file extension the RIB Properties Editor will use when creating a backup copy of the rib.properties on the server. This will be displayed in a dialog that appears on saving the file. The user can modify the extension of the backup file to whatever they choose before the file is saved.

3 Edit the gui.servlet.properties file in WEB-INF/classes.

```
##############################################
# GUI Project Variables
GUI.jdbc.driver=oracle.jdbc.driver.OracleDriver
GUI.rib.properties.SessionTimeout=900
GUI.rib.properties.local.FilePath=
```

- GUI.jdbc.driver should be set to the driver used to log in to the database for the main Portal login. The default driver that is contained the gui.war is an Oracle database driver.
- GUI.rib.properties.SessionTimeout should be set to the amount of time in which a session is timed out after being idle. The index.jsp will set the `HttpSession.setMaxInactiveInterval();` The default is 900 seconds (15 minutes).
- GUI.rib.properties.local.FilePath should be set to the directory where the RIB Properties should locally save the file while editing it. The default is to set this to <appserver-installation-directory>/<installed-application-name>/temp/, but can be changed to any directory on the application server.

Accessing the RIB Administration Tool

The RIB Administration Tool starts with the Main Portal screen. All access is performed using a Web Browser such as the Microsoft® Internet Explorer. The Web Browser downloads a Java applet from the application server.

Main Portal Screen

To access the Main Portal Screen, first Bring up the RIB Administration Tool from your browser by the following URL:

http://<hostname>:<port>/<installed-app-name>

where

<hostname> is the name of the host containing the application server, <port> is the port number used to access the application server <installed-app-name> is the name of the application the RIB Administration Tool has been installed under. The default is “gui”.

A login screen will appear. Enter in the login to the Hospital database you want to access using the Hospital Administration GUI. Even though the Message Statistics GUI and RIB Properties Editor GUI do not use a database connection, the RIB Administration tool uses the Hospital database login for authentication. This database login will need to be entered in to access the RIB Administration Tool regardless of whether the Hospital Administration GUI will be used.

Once logged in to the RIB Administration Tool, an index screen will appear containing links to the three applets: Hospital Administration GUI, RIB Properties GUI and Message Statistics GUI.

The login to the RIB Administration Tool will expire after the timeout set in the gui.servlet.properties. The user will be forced to login again if idle for the time set in this timeout property.

Hospital Administration GUI Applet

This applet contains the same functionality as the Hospital Administration Application detailed earlier in this manual. The only difference is the lack of a login window, since this login is derived from the main portal login.

See the help located in the Applet or Chapter 7 for more information about how to run this applet.

Message Statistics GUI Applet

This applet contains the same functionality of the RibTimings detailed in Chapter 16, but is now available in a GUI format. On loading of the applet, a window will appear with the following fields:

Filename: The default path and filename set in the gui.properties will appear here if they have been set. Otherwise, enter the full path to the timings log file located on the application server.

Status: Select the status of the statistics to display. All selects all the statistics available in the timings log file.

Interval: Enter the interval of time in seconds to create a bucket of timings. The RibTimings will group timings into a bucket to gather statistics. The default is 3600.

Start time: Enter the timestamp time in which the RibTimings should begin gathering statistics. The timings will not be gathered for any timestamp that is before the time entered in this field.

End time: This is not available unless a start time has been entered. The RibTimings will stop processing timings if it encounters any timestamp after the end time entered in this field.

Help Menu: This will display help on how to run the applet.

The main Statistics window will appear after selecting the **Ok** button. There are three areas to the main Statistics window: the message type list, the time period list and the statistics table. Select a message type from the list, and the time period list will display the time periods available for that message type. Once a time period is selected, the statistics table will display the corresponding statistics.

See the Help menu in the Message Statistics GUI Applet or Chapter 16 for more information on how to run this applet.

RIB Properties Editor

The RIB Properties Editor is a file editor that can be used for editing a file on a server using FTP. The file is copied to a local directory on the application server, and on save is copied back to the original server using FTP. A backup of the old file is created when the changes are saved.

This applet contains two windows: a Connection Window and a Main Menu.

Connection Window

The FTP connection information is entered on this window. The window appears on startup of the applet and by selecting

Open from the Main Menu. The following information must be entered:

File Name: The full path to the file on the server.

Server Name: The name of the server for the FTP Connection.

FTP User Name: The username for the FTP Connection.

FTP Password: The password for the FTP Connection.

Main Menu

This window contains the main actions for downloading and uploading a new RIB Properties file. The actions available are:

Open: Displays the connection window for retrieving the file.

Save: Saves the changes to the file back to the server. Displays a dialog in which a backup file extension can be entered; the default is displayed based on the gui.properties value.

Cancel: Cancels changes to the file. A dialog is displayed to verify that all changes should be discarded.

Exit: Exits from the applet.

Files and classes contained in the war file

Classes:

com.retek.rib.gui.AppletCoder: used for encoding and decoding information sent from applets to servlets

com.retek.rib.gui.HospitalUIApplet: main Hospital Administration class, contains all applet GUI code

com.retek.rib.gui.HospitalUIHelper: Hospital Administration class, contains calls to servlet

com.retek.rib.gui.PropertiesUI: main RIB Properties Editor class, contains all applet GUI Code

com.retek.rib.gui.PropsHelper: RIB Properties Editor class, contains calls to servlet

com.retek.rib.gui.StatisticsUI: main Message Statistics class, contains all applet GUI code

com.retek.rib.gui.StatsDBHelper: Message Statistics class, contains TableModel implementation

com.retek.rib.gui.StatsHelper: Message Statistics class, contains calls to servlet

com.retek.rib.gui.TableMap and **com.retek.rib.gui.TableSorter:** classes used for TableModel implementation for both applets

com.retek.rib.gui.DBConnection: used by index.jsp to test authentication with main RIB Administration login

com.retek.rib.gui.HospitalUIDBHelper: Hospital Administration class, contains TableModel implementation and command calls

com.retek.rib.gui.HospitalUIServlet: Hospital Administration servlet class

com.retek.rib.gui.PropertiesServlet: RIB Properties Editor servlet class

com.retek.rib.gui.TimingsServlet: Message Statistics servlet class

Other files:

js/apps.js: javascript file for RIB Administration index page

taglibs/gui.tld: tag library for RIB Administration index page

WEB-INF/lib/classes12.jar: contains Oracle Database Driver

WEB-INF/lib/retek-rib-support.jar: contains base code for Hospital Administration and Message Statistics functionality

WEB-INF/lib/retek-sbyn.jar: contains base code for Hospital Administration

WEB-INF/lib/etdRibMessages.jar: contains base code for Hospital Administration

WEB-INF/lib/stcjs.jar: contains base code for Hospital Administration

WEB-INF/web.XML: contains servlet mappings and session defaults

WEB-INF/classes/gui.servlet.properties and **gui.properties**: properties files used by RIB Administration Tool and applets

WEB-INF/classes/rib.properties: properties file used for Hospital Administration

HospitalUIHelp.html, StatisticsHelp.html and **PropertiesUIHelp.html**: help files for the applets, displayed by selecting Contents from the applet's Help Menu.

errorpage.jsp: error page for RIB Adminstration index and login pages

index.jsp: main index page for RIB Administration

login.jsp: main login page for RIB Administration

HospitalUI_en_US.properties, PropertiesUI_en_US.properties and StatisticsUI_en_US.properties: properties files containing GUI text for internationalization purposes

Chapter 9 – Message Statistics Command Line Utility

Overview

The Retek Integration Bus (RIB) logs set of timing entries whenever it creates, transform, routes, filters, or subscribes to messages on the RIB. These time entries are then post-processed by some other means to roll-up the data. This method was deployed to create a standard set of classes to perform this rollup and to create an internal summary storage of rolled up statistics. The displaying of the rolled up information is done via writing to the timings file or via message publication collaboration. You can then use this information to determine if the system is functioning correctly or if an application problem exists.

The same classes are used for this implementation as in the Administrative GUI applet.

Requirement

The following classes need to be deployed in order to gather the timings statistics. BucketSet.java, BucketTimingsMain.java, ProcessTimingsLog.java, RibFileLogger, RibTimings.java, StatsBucket.java, TimestampType.java, TimingsBucket.java, TimingsLog.java and TimingsType.java.

Description of the classes

The BucketSet contains a set of TimingsBuckets for a specific period of time. Each TimingsBucket contains a statistical rollup of timings for a specific Timing Type, Message Family, Message Type, and Processing Status combination. For some processing statuses, the Message Family and Message Type may be null. Additional BucketSet objects may be derived from an initial BucketSet object that contain some subset filetered by Timing Type, Message Family, Message Type and Processing Status. This class should be the interface to create or retrieve a specific bucket based on a combination of the identification fields – BucketSet Name, threshold, timings interval length, interval number.

A StatsBucket object is the holder of statistical information. From a StatsBucket, you can retrieve the average interval time, the standard deviation (n-1), the minimum time, the maximum time, the number of times the time is above a certain threshold value, the threshold value used (a constructor parameter), and the average value that exceeded the threshold. Every call to the StatsBucket.update() method results in these updated statistics.

A TimingsBucket object is a StatsBucket associated with a TimingType, Message Family, Message Type, and Processing Status. Multiple TimingsBuckets can be rolled up into a single StatsBucket.

The TimingsLog class is designed to read a file containing time stamp log entries and create a bucket array from the data. You can then manipulate or display this data as needed.

The RibTimings class is a wrapper around all the Timings Statistics class to produce a report through a User Interface.

Prerequisites to run the Timings Statistics:

The rib.properties should have all the properties defined for the e*Ways to get the timings statistics. The command line arguments to run the RibTimings and BucketTimingsMain class is:

java RibTimings filename status [CSV] [internal [time | all]]

java BucketTimingsMain filename status [internal [time | all]]

Where status is SUCCESS, FAILURE or ALL (case insensitive),

Interval is seconds for each report,

Time is in the form HH: MI: SS and only the interval containing the time is reported.

Note that the retek-rib-support.jar should have all the Timings statistics class within it. The retek-rib-support.jar should be placed to the correct CLASSPATH. Usage: java –cp –classpath < retek-rib-support.jar>

How the output appears to be when Timings Statistics report is run:

ewReceivingToRMS.timings SUCCESS 84400

Timings for the Period 00:00:00 to 23:26:39

Timing	Count	Average	STD Dev	Time Sum	Time^2 Sum	Min	Max	Threshold	Over Threshold Count	Over Threshold Sum	Over Threshold Avg
SUB_BETWEEN_COLLAB	999	0.07885	0.00099	78.772	7.18824	0.061	0.554	10	0	0	?
SUB_B4_CONSUME	1000	0.0114	0.00029	11.405	0.21223	0.009	0.265	10	0	0	?
SUB_IN_CONSUME	1000	0.14611	0.00084	146.111	22.06014	0.133	0.609	10	0	0	?
SUB_AFTER_CONSUME	1000	0.00845	0.00005	8.455	0.07404	0.007	0.024	10	0	0	?
SUB_TOTAL_IN_COLLAB	1000	0.16875	0.00093	168.751	29.3368	0.153	0.63	10	0	0	?

Chapter 10 – RMS Batch Message Program

Overview

The RmsBatchMsg class allows a user to run RMS 9 publishers and subscribers without the use of Seebeyond. The RmsBatchMsg class will export RMS publishing data into XML files, and import XML file data into RMS subscribers.

Running RmsBatchMsg

A properties file has been created for each publishing and subscribing API released in RMS 9.0.15. The following is a list of RMS 9.0.15 APIs and their corresponding properties files:

Publishing APIs

API	properties file
RMSMFM_COSTZNGRP	costzngrp.properties
RMSMFM_LOCLIST	loclist.properties
RMSMFM_ORGHIER	orghier.properties
RMSMFM_PARTNER	partner.properties
RMSMFM_SEASON	season.properties
RMSMFM_STORE	store.properties
RMSMFM_SUPPLIER	supplier.properties
RMSMFM_WH	wh.properties

Subscribing APIs

API	properties file
RMSSUB_ITEMLIST	itemlist.properties
RMSSUB_XITEM	xitem.properties
RMSSUB_XITEMLOC	xitemloc.properties
RMSSUB_XUDA	xuda.properties

Before running an API, make sure the following jars are in the CLASSPATH:

- rmsbatch.jar
- retek-rib-support.jar
- retek-pub-trans.jar
- log4j.jar
- xmlParserAPIs.jar
- classes12.jar
- xerces_1_2_3.jar

To run an API, simply run the RmsBatchMsg class with the API's properties file.

Example:

```
java RmsBatchMsg costzngrp.properties
```

Properties files

Properties - common

Each properties file has the following editable properties

rmsbatch.RmsBatchMsg.message_nodes_in_commit=1

```
rmsbatch.<<subclass
name>>.url=jdbc:oracle:thin:@<<machine>>:<<port>>:<<db>>
rmsbatch.<<subclass name>>.user=user
rmsbatch.<<subclass name>>.password=password
rmsbatch.<<subclass name>>.package_name=<<RMS package>>
rmsbatch.<<subclass name>>.message_family=<<message family>>
```

The <<subclass name>> is set one of the following in the properties file:

- OracleObjectGetNxtSource - for publishing APIs that use Oracle Objects
- ClobGetNxtSource - for publishing APIs that use CLOBs
- OracleObjectConsumeDestination - for subscribing APIs (all subscribing APIs in RMS 9.0.15 use Oracle Objects)

message_nodes_in_commit - specifies the number of messages processed between each commit.

NOTE: This does NOT specify how many messages to publish from an API. ALL messages will be published from the specified API in one call to RmsBatchMsg.

url - specifies the location of the RMS database. The <<machine>>, <<port>>, and <<db>> need to be filled in for each properties file.

user - the username for logging into the RMS database.

password - the password for logging into the RMS database.

package_name - the name of the RMS API being called. In the costzngrp.properties example, this is set to RMSMFM_COSTZNGRP.

message_family - the name of the message family. In the costzngrp.properties example, this is set to COSTZNGRP.

Properties - publisher-specific

```
rmsbatch.RibMessagesFileDestination.file_name=<<filename>>
```

file_name - specifies the file to write messages to.

NOTE: The program will overwrite any existing file without warning, so make sure to move an existing output file before running the program again.

Properties - subscriber-specific

```
rmsbatch.OracleObjectConsumeDestination.type_map.<<message
type>>=<<oracle object type>>


rmsbatch.RibMessagesFileSource.file_name=<<filename>>
```

type_map - specifies which oracle object type is used for each message type in the family. There should be one property for each message type. For example, the itemlist family has 5 message types, “itemlistcre”, “itemlistmod”, “itemlistdel”, “itemlistdtlcre”, and “itemlistdtldel”. Here are the type_map settings for the itemlist family:

```
rmsbatch.OracleObjectConsumeDestination.type_map.ITEMLISTCRE=RIB_ITE
MLISTDESC_REC

rmsbatch.OracleObjectConsumeDestination.type_map.ITEMLISTMOD=RIB_ITE
MLISTDESC_REC

rmsbatch.OracleObjectConsumeDestination.type_map.ITEMLISTDEL=RIB_ITE
MLISTREF_REC

rmsbatch.OracleObjectConsumeDestination.type_map.ITEMLISTDTLCRE=RIB_
ITEMLISTDESC_REC

rmsbatch.OracleObjectConsumeDestination.type_map.ITEMLISTDTLDEL=RIB_
ITEMLISTREF_REC
```

file_name - specifies the XML file to read messages from.

Chapter 11 – Multi-Thread feature for the e*Ways

What is a Thread?

A thread (sometimes called an *execution context* or a *lightweight process)* is a single sequential flow of control within a program. The threads are used to isolate tasks.

Amdahl's Law

Assuming that an application is multithreaded (programs written to execute in a parallel manner, rather than a serial or purely sequential one), there are inherent difficulties in making a program run faster proportional to the number of processors: the program needs to be written in a parallel fashion, and the program itself must be resource friendly.

Amdahl's Law explains this**:** "…the performance improvement to be gained from using some faster mode of execution is limited by the fraction of the time the faster mode can be used." This law applies to more than just changing sequential code to parallel code.

Assume that a program consists of two main parts, A and B, and that each can be optimized. Part A represents 90% of the execution time, and B represents the remaining 10%. Assume that B can be optimized in such a fashion so as to be able to finish in one tenth the time of the original version, and that A can be optimized so as to complete its part of the program in 2/3 the time it previously needed. If both parts A and B take the same amount of time to be optimized, and the programmer has time only to optimize one part, which should the programmer work on? Obviously, he/she should work on part A.

Assume that this program requires 100 seconds to complete. Part A consumes 90 seconds of execution time, and B requires 10 seconds of execution time. After optimization Part A would take 60 seconds, and B a mere 1 second. The choice is between a total of 70 seconds of execution time if A is optimized, and 91 seconds if B is optimized.

Multi-threaded feature for Subscriber, TAFR and Publisher:

What are the situations where multi-threading can help?

Multi-thread can be a valuable tool to increase performance, but it does not help in every situation. First and foremost, there is some overhead associated with multi-threading. Therefore, multi-threading should not be used unless a performance problem exists. If you have an e*Way that is processing only several messages per minute, this would probably not be a good candidate for multi-threading. This is because you would be increasing the overhead on the server, but you would not get any benefit from that increased overhead. A good candidate for multi-threading would be an e*Way that continually receives a stream of multiple messages per second, or that receives bursts of many messages within a short period of time. One example might be an e*Way that receives real-time updates from time to time, and also receives periodic batch updates consisting of a large number of updates.

Multi-threading still may not help the above situation unless the server has multiple processors to share the load. If the machines has only a single processor, the additional overhead associated with switching between multiple threads may actual slow the processing of messages down. If the threads can be doled out to separate processors, that is where performance can really be enhanced.

When multi-threading is used, it should be used across all the e*Ways that process messages for a message family. That would include publisher, subscriber, and TAFR e*Ways. It would not be helpful to have a publisher sending messages very quickly and efficiently, but if the subscriber can process them only so fast, the bottleneck will exist in the subscriber e*Ways.

The Subscribing, TAFR and Publishing e*Ways provides the multi-threading features together with the Publishing e*Way. In order to incorporate this feature, there is a certain step that needs to be followed. The following classes cater the multi-thread features for the e*ways - HospitalController.java, HospitalRetryController.java, RibCollabController.java, RibProperties.java, and MultiThreadUtil.java.

Go to the SeeBeyond e*Gate Enterprise Manager -> select the e*Way which needs to run the Multi-thread feature and copy the collaboration and paste the number of times it needs to be multi-threaded. Rename the collaboration so that the e*Way has unique identification of the multi-thread collaboration. If there is 4 Publishing e*Ways with multi-thread features, then there should be 4 Subscribing e*Ways and as a result, there should be no thread number greater than 4.

The Retry feature has been enhanced with the Multi-thread features. The rib.properties file needs to have the following entries:

a In the multi-threading properties section, there should be an entry for each family name and total number of threads implemented, e.g.

Mfm.messageFamilyName.total_threads=n

mfm.Alloc.total_threads=4

mfm.Alloc.colAllocFromRMS_1.thread_num=1

mfm.Alloc.colAllocFromRMS_2.thread_num=2

mfm.Alloc.colAllocFromRMS_3.thread_num=3

mfm.Alloc.colAllocFromRMS_4.thread_num=4

When a multi-threaded e*Way comes online, the system will check this value for each individual collaboration as it comes online. As the collaboration comes online, the system keeps track of how many have come online so far. If the number specified in the rib.properties entry is exceeded, a runtime exception will occur.

b The e*Way specific logging level verbose should be set to 'Y' for the e*Way which needs to be run for the multi-thread feature, e.g. log.ewAllocFromRMS.verbose=Y

All collaboration have different database connection settings for the HospitalRetry. If one decides to have multi-thread based queues, we suggest you set-up hospital retry queues. Each application should have its own collaboration in the hospital e*Way – ewHospitalRetry.

c The next step is the replication of the publishing and subscribing event types. Assume our original event type is named, "etTestMessageType". Since our total threads is four, we want to make three copies and then rename them. As mentioned above, there are naming conventions that you need to follow. Each event type needs to have to end with an underscore and a unique digit. In this case, we will name the event types, "etTestMessageType_1", "etTestMessageType_2", "etTestMessageType_3", and "etTestMessageType_4". It has to end with an underscore and a sequence of digits.

d Next, replicate the collaborations for the respective subscribing and publishing e*Ways. Before we do this, though, we should go into our original collaboration and verify that the publishing event type has been automatically renamed as the new name for our original event type. For example, it should be as follows;

Event Type	**Corresponding Collaboration**
etTestMessageType_1	colTestSubCollaboration_1
etTestMessageType_2	colTestSubCollaboration_2
etTestMessageType_3	colTestSubCollaboration_3
etTestMessageType_4	colTestSubCollaboration_4

In renaming all the event types and corresponding collaboration, the system automatically publishes events to the correct topics.

If you created a new connection point and selected the properties with e*Way Connection Type as ‘SeeBeyond JMS’, the “New” button is enabled for the ‘e*Way Connection Configuration File’. After pressing the “New” button, it displays two options. Selecting the “Internal: Connect to JMS IQ Mgr within this schema” and JMS IQ Manager as ‘iqmJMS’, it sets the configuration file. Click the ‘Edit’ button and go to the ‘Goto Section’ for ‘General Settings’ and select the ‘Goto Parameter’ for ‘Message Selector’. You need to add within this ’Message Selector’ like ‘Thread_Value=1’. This message selector is used for subscription. The Java class programs cater this piece of information and as a result, this feature does not need to be set in the connection point of SeeBeyond e*Gate Enterprise Manager.

Before ‘Start’ of any e*Ways to run the multi-thread feature, log onto SeeBeyond e*Gate Monitor for the respective schema, then click the ‘Launch JMS Administrator’ button to open the ‘JMS Administrator’ window. On expanding the ‘iqmJMS’ option, the ‘Topics’ would be displayed. Select the event type that needs to be run for the multi-thread feature and check out whether any collaboration for the subscriber is associated with the event type. Delete any collaboration to the subscriber by selecting the collaboration. Press the right-mouse-button and select ‘Delete Subscriber’. Once this process is completed, start the e*Ways from the e*Gate Monitor and run the multi-thread feature.

The RIB_MESSAGE table has thread_value field, which collects the multi-thread information. The MultiThreadUtil class has the NumThreads and ThreadValue properties defined for Multi-threading.

Chapter 12 – Troubleshooting

SeeBeyond Platform

This section lists a general approach to troubleshooting problems.

If a problem persists, information can be obtained by turning on e*Way logging and tracing. For information on this, see the Error, Trace, Debug Log Files section of Chapter 5.

Problems starting a RIB component

A RIB adapter may not start or can terminate soon after it has started. There are two possible sources of this problem: incorrect configurations and environment problems.

Incorrect configurations

Many problems can arise that involve improper or incorrect configuration file or properties:

- **Connection Point Names:** If a Connection Point is renamed or deleted, then any collaboration that references it will have errors and will not be able to process data.
- **Oracle Connection Point User Names and Passwords:** Incorrect specification of the Database Server, System ID (SID), User Name or User password will result in errors for all adapters using the connection point. Note that the user password is stored as an encrypted string.
- **JMS Connection Point TCP/IP Address:** JMS Connection Ports must specify the correct TCP port number and IP address or host name. A common problem that may occur when migrating a schema from one environment (such as a development environment) to another (such as a test environment) is that these are not changed. The configuration files for this contain ASCII characters. Retek recommends creating scripts to modify these values when migrating the RIB between development, test, and production environments.

Environment problems

Some problems starting adapters are the result of environment or system errors.

- **Registry or Control Broker not started: The SeeBeyond EAI system does not automatically start up the host registry daemon or** any of the control brokers found within a schema. For Unix Systems, these must be started via a startup script, typically upon system boot. On Microsoft Windows systems, these are typically installed as services and should be started automatically. There must be one control broker per host per schema found in the registry.
- **JMS IQ Manager NOT started:** The RIB adapters that use a JMS Connection Point require that the JMS IQ Manager be up and running before any adapter can access it.
- **XA Transaction Logs deleted:** Never delete the XA transaction logs or you risk losing data on the JMS queues, losing data associated with prepared transactions in Oracle, or having many other problems. Oracle prepared transaction IDs can be found in the DBA_2PC_PENDING view. SeeBeyond transaction logs are found beneath the directory <EHOME>/client/XALogs.
- **XA Not installed in Oracle:** An adapter can have problems starting if the XA package and libraries are not installed in the Oracle database.
- **JMS IQ Manager Directory specified via a relative pathname:** This becomes a problem if the control broker is started from a different directory than usual. As a rule, always use a fully qualified directory name.
- **Multiple Duplicate Control Brokers:** On Unix systems, the stccb command must be executed once per control broker. If multiple identical stccb commands are issued, components chaos may ensue. The Unix command “ps –ef | grep stccb” lists running stccb processes. Use the “kill” command to bring down the extra stccb process
- **SYS.DBMS_PICKLER ERRORS from Oracle:** Usually occurs because the user used in the connection point does not have sufficient priviledges to the Package or RIB Objects being referred to in the application. Either change the user that is being used or make sure proper permissions and synonyms are created in Oracle.

Invalid JMS selectors

This section applies to the following message that may appear in the RIB Log file for an adapter or e*Way:

```
Current Message Selector = ‘’ but it should be
= 'threadValue=’1’ and (retryLocation is null or retryLocation =
‘<eWayName>.<collaborationName>’)’

There are up to <some number> messages awaiting processing by this
subscriber");

To fix this problem Export all messages on Topic and delete the
subscriber with the following command: stcmsctrlutil –host …
```

Where <eWayName> is the name of the e*Way, <collaborationName> is the name of a collaboration, and <some number> is a number.

In order to insure exactly once processing, RIB adapters use JMS Selectors to filter out messages that are specific to a single subscriber when multiple subscribers go against the same JMS Topic. The selector will insure that only the correct subscriber will get a message re-posted from an Error Hospital. In a multi-threaded environment, selectors are used to insure that each subscribing thread receives the correct stream of messages when sharing a JMS topic.

In order to simplify configuration, the selector is determined programmatically at startup. Unfortunately, when a JMS server is booted, SeeBeyond dynamically checks its registry for the JMS selector used by e*Way connection points. When the JMS is booted, it creates a JMS durable subscriber using the value from the registry, not from a previous instance of the JMS. When this occurs, the JMS durable subscribers are re-created with empty or blank selectors. At this time, Retek is working with SeeBeyond to change this behavior. As of the 10.3.2 release, an ESR has been made available to disable this behavior when the JMS is booted. Check with Retek customer support for more information.

A RIB Properties file property, **default.MessageSelectorCheck,** determines whether the e*Ways should check if the correct selector is in place. If set to true, the following is performed when the e*Way is started:

1 During the call to userInitialize(), the e*Way examines the JMS Topic it subscribes to.

2 The e*Way verifies its Durable Subscriber contains the correct selector. If the selector is missing or incorrect **and there are no messages queued for the subscriber,** the Durable Subscriber is deleted and re-created with the correct JMS Selector.

3 If messages are queued on the JMS Topic for an invalid durable subscriber, the e*Way is shut down and the error mentioned above logged to the e*Way's RIB Log file.

If an e*Way is shutdown due to an invalid selector, the following process can fix the situation:

a Shut down any message publishers for the messages handled by the TAFR or subscribing adaptor.

b Edit the rib.properties file, change **default.MessageSelectorCheck** from "true" to "false".

c Bring up the e*Way and wait for it to process all messages on the topic.

d Bring down the e*Way. Change **default.MessageSelectorCheck** back to "true".

e Bring up the e*Way again. The selector should now be valid.

To avoid this problem, always try to perform the following:

1 Always bring up message subscribers before message publishers.

2 If at all possible, always turn off messages publishers and wait for all messages to drain before shutting down the JMS server.

In the RIB 10.3.2 release, two new scripts, *start_rib* and *stop_rib*, are available to bring up or down the RIB schema in a controlled sequential manner. These scripts use a configuraton file that details which e*Ways should be brought up and the order this is done. A switch is available to specify an implementation specific configuration file.

Publisher being shutdown during initialization

If **default.SubscriberCheck** is set to true or non-existent (default action), the publisher is configured to verify that a subscriber does exist for each topic it will publish to. If at least one subscriber is not found, the publisher will shutdown and an exception will be thrown, causing the collaboration to stop. During initialization this causes the e*Way to shutdown.

The solution is to verify that a subscriber does exist for the topic that the publisher will be publishing to. Turning on verbose logging will aid in tracking down which topic it will publish to.

Message processing problems

This section describes possible problems the RIB might occur processing messages. It gives a brief description of the problem symptoms and suggested actions.

Messages "disappear" when published by a non-Retek application

Description: A non-Retek standard adapter publishes messages successfully, but they appear to vanish and none are delivered to the Retek adapter.

Action: Many times this is due to the messages not containing the correct JMS message properties. All messages must contain a message property named *threadValue*. By default, RIB adapters select only those messages with a *threadValue* of '1'. Hence, have the publisher create and set a JMS Message Property named 'threadValue' with a value of '1'.

A non-Retek application recieves messages being re-tried from the Error Hospital that it had already successfully processed.

Description: A non-Retek is delivered messages successfully consumed by itself but were not successfully processed by another subscriber. When the message is retried from the Error Hospital both subscribers reprocess the message.

Action: Insure that the subscriber uses a selector that checks the *retryLocation* JMS Message Property. All messages published from the Error Hospital to a JMS Topic for retrying will contain a value of *retryLocation* specific to one and only subscriber to actually perform the retry processing. A typical JMS selector is the following:

threadValue ='1' and (retryLocation is null or retryLocation ='<locationID>')

where <locationID> specifies the adapter thread to perform the retry processing.

No messages processed

Description: An adapter is not able to update the Error Hospital, publish new messages, or successfully process messages from a queue if the XA package is not installed and/or activated in the Oracle database. No messages leave the RIB queue, since XA is required for inserting messages into the Error Hospital.

Action: Install the XA libraries and packages.

Publishing adapter hangs

Description: Some messages were published before, but now no messages can be published at all. The publishing e*Way hangs whenever it tries to send a message to the JMS queue.

Action: The JMS queue may be full. This could be due to problems with subscribing e*Ways. For example, the database the subscriber is connected to does not have the Oracle XA libraries installed. Check to make sure that subscribers can be started successfully and, if possible, have no errors processing messages.

This problem can also be caused by an e*Way that is designed to connect to an application that is not installed. Messages remain in the JMS queue for all adapters it believes will, in some future time, pull off messages. The standard RIB schema contains all adapters for all Retek applications. Delete any e*Way that is not brought up as part of your version of the RIB schema.

XA lock(s) cause problems with one or more messages

Description: Database locks are normally held within a 2-phase commit operation transaction until the second phase has started or a rollback is issued. If a system failure has occurred between the end of the first phase and the beginning of the second phase, then these locks are held forever, unless administrative actions are taken.

The following Oracle message may appear in the logs when this occurs:

```
ORA-01591: lock held by in-doubt distributed transaction <XID>
```

where <XID> is a string of three numbers separated by periods (such as 1.21.17).

Action: If possible, fix the problem and display the e*Way associated with the transaction. The e*Way recovery process should complete the transaction and remove the lock. If this cannot occur, evaluate whether the transaction should be committed or rolled back administratively.

The following procedure commits the Oracle part of a transaction:

Note: This process risks a “Heuristically Mixed” transaction status: the Oracle work in a transaction committed, but the SeeBeyond work rolled back. Careful analysis should always be performed before attempting to perform this procedure.

1 Determine the Global Transaction ID (XID) of the transaction to be committed. All prepared transactions will have an entry in the DBA_2PC_PENDING view. With SeeBeyond e*Gate, the XID is a string of three period-separated numbers (such as 123.45.890). This view requires administrator privileges to access its contents.

2 Issue the following SQL, using a facility such as SQLPLUS:

```
COMMIT FORCE ‘<XID>’ ;
```

where <XID> is the XID of the transaction. Successful execution of this command requires administrator privileges that are not granted to most users.

3 Or, commit the work using the following SQL:

```
ROLLBACK FORCE ‘<XID>’
```

This has the same condition as forcing a commit. That is, the Oracle work rolled back and the SeeBeyond work committed.

User defined alerts are displayed

Description: The e*Gate Monitor reports many “User Defined Alerts”. This results from trying messages in the Error Hospital too many times.

Action: If possible, determine the root cause. These messages may be going into the Error Hospital due to a field value found in the publisher but not found in the subscriber. Examine the messages in the error hospital and check to see what the error is. If nothing is apparent, turn on trace logging in the e*Way and look at the log file for more information. These alerts might also be due cross message family dependencies, so verify that all appropriate publishing and subscribing adapters are up and running.

Once the problem has been fixed, increase the Max attempts for all of the messages in the error hospital so that they will be republished. Otherwise, the data contained in these messages will never be processed again. Furthermore, any subsequent messages referencing the same business entity (such as the same Purchase Order) will be held in the Error Hospital as well.

Messages not getting to the correct subscriber

Description: The TAFR routing functionality appears to be malfunctioning. Messages go to the wrong subscriber.

Action: Examine the rib.properties file used. Verify that lines exist in this file for all locations and that the translation of the <facility_type>.<facility_code> is correct.

TAFR not processing any messages

Description: The TAFR is not processing any messages.

Action: Examine the rib.properties file used. Verify that lines exist in this file for all locations and that the translation of the <facility_type>.<facility_code> is correct. Using the e*Gate Monitor application, verify that the JMS server (the JMS IQ Manager) used as the destination for the messages is running. Look for any alerts published from the TAFR adapter.

TAFR being shutdown during runtime

Description: If **default.SubscriberCheck** is set to true or non-existent (default action), TAFR’s are configured to verify that a subscriber does exist for each topic it is publishing to. If no subscriber is found, an exception will be thrown, causing the collaboration to stop. This will cause the e*Way to shutdown also.

Action: Verify that a subscriber does exist for the topic that the TAFR is publishing to. Turning on verbose logging will aid in tracking down which topic it is publishing to.

Shutdown problems

An adapter or IQ Manager will not shutdown unless it is between messages. Once a shutdown command has been accepted by a component, it will not accept new work. However, existing messages will still be processed.

In rare circumstances, it may be necessary to manually "kill" an adapter because a message processing thread is held due to a database lock or other resource contention conflict. If this occurs, you can kill the process using the Unix "kill" command or, for Microsoft Windows platforms, the task manager.

Note: If the RIB Installation Instructions were followed, a "plist" script will exist in the $EHOME directory which displays all current processes.

Because of the distributed nature of the e*Gate platform, manually issuing kill commands for the control broker process (stccb) is not recommended unless all attempts to shutdown the control broker using the e*Gate Monitor application has failed.

Hospital admin GUI and command line utility

There are two types of problems using the Hospital admin GUI or command line interface: Java class instantiation problems and Database connection problems.

Java class instantiation problems

Most Java class instantiation problems involve the inability to create a java class because it doesn't know where the class definition is. Typically, an incorrect CLASSPATH environment variable is the cause. The scripts `hospital`, `querymsg`, `readmsg`, `deletemsg`, `updatemsg`, and `stopmsg` all source the `hospital-admin.env` file to set the correct class path. This file assumes that the directory <EHOME>/client/classes exists and contains the required JAR files. However, there are some circumstances where needed jar files do not exist here. The main scenario where this can occur is before any RIB e*Way has been started that requires the specific JAR file. Listed below are some JAR and ZIP files needed, and alternative locations:

- **xalan.jar** – needed for reading message contents. The JAR file contains the definition of the class **org/XML/sax/ContentHandler.** This JAR file can also be found in the "server" directory of the e*Gate installation: <EHOME>/server/registry/repository/default/ThirdParty/RSA/certj_2.0.1/classes/xalan.jar
- **classes12.zip** – needed for the JDBC driver to connect to the Oracle9i database. This file is normally found in <EHOME>/client/ThirdParty/oracle/classes/classes12.zip. It may also be downloaded from the Oracle Technology Network website. See http://otn.oracle.com/software/content.html for more details.
- **retek-rib-support.jar**
  etdRibMessageEnvelope.jar
  stcjcs.jar – these JAR files are used by the Error Hospital should be in <EHOME>/client/ directory tree. Alternate copies of these files are found in the <EHOME>/server/repository directory tree.

Database connection problems

An inability to connect to the database may be due to a missing JDBC driver code. The file classes12.zip should be present in the CLASSPATH and exist on the local machine where the utility executes.

Other possible connection problems include:

- Bad username/password/SID specification in the `hospital-admin.properties` file or a missing hospital-admin.properties file.
- A connection will not be made if using a PC to execute the utility that is located outside of a firewall that is not configured to accept connections to the database.

J2EE Platform

This section lists a general approach to troubleshooting problems using WebSphere as the application server.

Available tools

The following are available for assisting with troubleshooting:

- SeeBeyond JMS Administrator
- SeeBeyond e*Way log files
- RIB Log files
- WebSphere server log files

Messages not getting consumed by application

Once messages are published to the RIB, and have made it through the appropriate TAFR e*Way, they should be immediately picked up by the WebSphere Application Server (Message-Driven Beans). If not, either there is an incorrect JMS configuration, or WebSphere's Message Listener Ports have lost connection to the SeeBeyond managed JMS queue.

Incorrect configurations

Within WebSphere, there are three things that must be correctly configured in order for messages to be consumed by the Message-Driven beans:

- **File System JNDI/Context file:** In the …/WebSphere/sbynjndi directory, there is a hidden file named **.bindings**. This file contains the actual serialized SeeBeyond JMS Objects. If this file doesn't exist or was created with a different JMS hostname/port combination, the Generic JMS Provider configuration will be invalid. Refer to the RCOM installation guide on how to create this file.
- **Generic JMS Provider:** If the JMS Connection Factory and Destinations are not properly configured, the listener ports will not be able to start.
- **Message Listener Ports:** Each Message Listener Port must be correctly configured with a valid Connection Factory and Destination. These are configured in the Generic JMS Provider area.

Lost connection to JMS

The following, would cause WebSphere to not "listen" to JMS:

- SeeBeyond's JMS Queue was not running when the Application Server was started, the Message Listener Ports would not be connected.
- If SeeBeyond's JMS Queue should happen to be stopped after the Message Listener Ports have successfully started.

In either case, the Application Server will have to be restarted after ensuring that SeeBeyond's JMS queue running.

Messages not getting published from application

Published messages should go directly into SeeBeyond JMS to be consumed by other e*Ways. The WebSphere server log file and the SeeBeyond JMS Administrator are the two tools to use for troubleshooting publishing messages from a J2EE application.

Incorrect configurations

Within WebSphere, there is one thing that must be correctly configured in order for messages to be published by the Publisher beans:

- **File System JNDI/Context file:** In the …/WebSphere/sbynjndi directory, there is a hidden file named **.bindings**. This file contains the actual serialized SeeBeyond JMS Objects. If this file doesn't exist or was created with a different JMS hostname/port combination, the Generic JMS Provider configuration will be invalid. Refer to the RCOM installation guide on how to create this file.

JMS Provider down

SeeBeyond's JMS queue must be running for the Publisher EJB to be able to publish messages. If this is not the case, ensure the JMS queue is running and try to publish again.

Error Messages

When troubleshooting using the log files, here are some of the things to look for, as well as some potential solutions:

Exception Class Name	Exception Message	Possible Solution
org.xml.sax.SAXException	Parsing Error : File "http://www.retek.com/dtd/rib/DiffDesc.dtd" not found.	There are two potential solutions to this error. The first is to correct the data in the rib_doctypes database table in the RMS database. This solution is valid only if there is a row in the table whose value in the doc_name column matches the dtd document name. In this example the document name is, "DiffDesc.dtd". In the case where we do have a matching row, the problem is most likely that the doc_type_url column has an invalid url. It must consist of an http server and port number that points to a directory containing the dtd document. The second potential solution is that, if there is not a matching record in the rib_doctypes table, the entry for the default DTD URL in the rib.properties file is missing, or invalid. Keep in mind that we are talking about the rib.properties file for the RMS publisher, not the Rib ISO integration. The property name for the default DTD URL is, "dtd_url.default". Again, the value must consist of an http server and port number that points to a directory containing the DTD document. An example of an entry for the DTD URL in the rib.properties file is: http://hostname:8100/dtd/
com.retek.binding.rib.exception.PublishException	"PUBLISH_FAILED"	The PublishException is returned from the RIBMessagePublisherEJB when a message cannot be published to either JMS or be inserted into the Hospital. Check that the JMS is running and there are no database problems with the RIB datasource in WebSphere.

Exception Class Name	Exception Message	Possible Solution
com.retek.binding.rib.exception. InjectorException	"INJECT_FAILED"	The InjectorException class can contains a nested exception returned from the RCOM application. Most likely this nested exception will be a java.sql.SQLException. If it is, it will likely indicate a null constraint violation, integrity constraint violation, or unique constraint violation.
com.retek.binding.rib.exception. RIBIntegrationException	COMMAND_ FACTORY_UNABLE_ TO_READ_ PAYLOAD_OR_ BINDING_ PROPERTY_FILES	Either the payload.properties, or binding.properties file is not on the application's class path. Check the rns.sh, node_rns.sh, and node.sh on Unix, or rns.bat, node_rns.bat, and node.bat on Windows. Check these files for the classpath set in them to make sure the directory, in which the payload.properties or binding.properties file is located, is in the classpath. Alternatively, put the properties file into a directory that is on the classpath.
com.retek.binding.rib.exception. RIBIntegrationException	COMMAND_ FACTORY_CANNOT_ INSTANTIATE_ PAYLOAD	The payload.properties file does not contain an entry whose property name matches the Rib message family and Rib message type key, extracted from the Rib message itself. Either the XML for the Rib message does not have the appropriate family and/or type, or the payload.properties file is missing an entry for the family and type. See the section, "Retek Binding Configuration Files" under "Chapter 4 – Configuration Files".
com.retek.binding.rib.exception. RIBIntegrationException	UNMARSHALING_ ERROR	There is either something wrong with the XML that is being unmarshalled into the payload object, or the payload object is out of date with respect to the DTD and XML schema, from which the payload object was generated.

ISO Platform

There are several log files that are important to troubleshooting the Rib ISO integration module. All of the log files mentioned below will be found in the standard ISO "log" directory.

On the Windows operating system the log files are found in the directory

```
<install_dir>\chelsea\serverWdws\retek\sim\log
```

On the Unix operating system the log files are found in the directory

```
<install_dir>/chelsea/serverUnix/retek/sim/log
```

In both cases, <install_dir> is the directory the ISO application has been installed into.

The first, and most important log files, are the files that are specific to each individual API, whether publishing or subscribing. By default, their names are <RibMessageFamily>messagingcomponent.log (all lowercase) for subscribers, and <RibMessageFamily>publisher.log (all lower case) for the publishers.

Examples of a publisher and a subscriber are:

```
asnoutpublisher.log
```

```
asnoutmessagingcomponent.log
```

In addition to these log files, there are two log files pertaining to the entire RIB container. These are the RIBContainer_*nnnnn*.out, and the RIBContainer_*nnnnn*.err files. Any messages written by either the ISO application, or the Rib integration module, to standard out go to the "RIBContainer_*nnnnn*.out" file, while messages written to standard error go to "RIBContainer_*nnnnn*.err". Most messages, however, will go to the individual log files for the publishers and subscribers. If you do not find the detailed information you are looking for in the individual publisher or subscriber log file, you might be able to find it in one of these two files.

When troubleshooting using the log files, here are some of the things to look for, as well as some potential solutions:

Exception Class Name	Exception Message	Possible Solution
org.xml.sax.SAXException	Parsing Error : File "http://www.retek.com/ dtd/rib/DiffDesc.dtd" not found.	There are two potential solutions to this error. The first is to correct the data in the rib_doctypes database table in the RMS database. This solution is valid only if there is a row in the table whose value in the doc_name column matches the dtd document name. In this example the document name is, "DiffDesc.dtd". In the case where we do have a matching row, the problem is most likely that the doc_type_url column has an invalid url. It must consist of an http server and port number that points to a directory containing the dtd document. The second potential solution is that, if there is not a matching record in the rib_doctypes table, the entry for the default DTD URL in the rib.properties file is missing, or invalid. Keep in mind that we are talking about the rib.properties file for the RMS publisher, not the Rib ISO integration. The property name for the default DTD URL is, "dtd_url.default". Again, the value must consist of an http server and port number that points to a directory containing the DTD document. An example of an entry for the DTD URL in the rib.properties file is: http://hostname:8100/dtd/
com.retek.binding.rib.exception. ApplicationMessageInjectionExceptio n	"CREATE_FAILED", "MODIFY_FAILED", or "DELETE_FAILED"	There was a problem in the ISO application. The ApplicationMessageInjection Exception class can contain a nested exception. Most likely this nested exception will be a java.sql.SQLException. If it is, it will likely indicate a null constraint violation, integrity constraint violation, or unique constraint violation.

Exception Class Name	Exception Message	Possible Solution
com.retek.binding.rib.exception. RIBIntegrationException	COMMAND_ FACTORY_UNABLE_ TO_READ_ INJECTOR_ PROPERTY_FILE	The injector.properties file is not on the application's class path. Check the rns.sh, node_rns.sh, and node.sh on Unix, or rns.bat, node_rns.bat, and node.bat on Windows. Check these files for the class path set in them to make sure the directory, in which the injector.properties file is located, is in the class path. Alternatively, put the inject.properties file into a directory that is on the class path.
com.retek.binding.rib.exception. RIBIntegrationException	COMMAND_ FACTORY_UNABLE_ TO_READ_ PAYLOAD_OR_ BINDING_ PROPERTY_FILES	Either the payload.properties, or binding.properties file is not on the application's class path. Check the rns.sh, node_rns.sh, and node.sh on Unix, or rns.bat, node_rns.bat, and node.bat on Windows. Check these files for the classpath set in them to make sure the directory, in which the payload.properties or binding.properties file is located, is in the classpath. Alternatively, put the properties file into a directory that is on the classpath.
com.retek.binding.rib.exception. RIBIntegrationException	COMMAND_ FACTORY_CANNOT_ INSTANTIATE_ INJECTOR	The injector.properties file does not contain an entry whose property name matches the Rib message family and Rib message type key, extracted from the Rib message itself. Either the XML for the Rib message does not have the appropriate family and/or type, or the injector.properties file is missing an entry for the family and type. See the section, "Retek Binding Configuration Files" under "Chapter 4 – Configuration Files".

Exception Class Name	Exception Message	Possible Solution
com.retek.binding.rib.exception. RIBIntegrationException	COMMAND_ FACTORY_CANNOT_ INSTANTIATE_ PAYLOAD	The payload.properties file does not contain an entry whose property name matches the Rib message family and Rib message type key, extracted from the Rib message itself. Either the XML for the Rib message does not have the appropriate family and/or type, or the payload.properties file is missing an entry for the family and type. See the section, “Retek Binding Configuration Files” under “Chapter 4 – Configuration Files”.
com.retek.binding.rib.exception. RIBIntegrationException	UNMARSHALING_ ERROR	There is either something wrong with the XML that is being unmarshalled into the payload object, or the payload object is out of date with respect to the DTD and XML schema, from which the payload object was generated.