Retail passion. Innovative solutions.

Retek® Integration Bus™ 11.1

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

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
Hong Kong	800 96 4262
Korea	00 308 13 1342
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 Components 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.

SeeBeyond components

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.

Registry

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, 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.

Schemas

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).

Control brokers and participating hosts

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

There is one control broker per “participating host” per SeeBeyond e*Gate schema. A participating host is a logical construct used. The control broker’s TCP/IP address and the participating host’s name are associated with each other within the registry.

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

Events and event type definitions

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

Collaborations

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

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

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

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

e*Ways

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. e*Ways contain one or more "Collaborations" that are triggered from some event. A collaboration works on a collection of input and output events, which may be messages going to or from queues, or passing to or from an application's RIB APIs.

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

The RIB uses a specific type of e*Way, the Java "Multi-mode" e*Way, which can function as both an external source or sink and an internal connector. The Multi-mode e*Way is a grouping of logical collaborations into a single physical process or program.

Intelligent Queues and JMS Intelligent Queues

Intelligent Queues (IQ) hold published messages and maintain a record of what subscribers have received the messages. Many types of Intelligent Queues either wrapper the message storage mechanism or bridge to another queuing system. The SeeBeyond e*Gate system installed with the RIB includes a Java Messaging Service (JMS) IQ. JMS Intelligent Queues are queues that may be accessed using the Java Message Service API.

IQ Managers and JMS IQ Managers

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

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

The RIB is designed to only retrieve and publish messages to a JMS compliant server. The preferred JMS implementation is the SeeBeyond's standard JMS implementation. As of the 10.3 release, Retek has not certified other JMS implementations or interfaces.

e*Way Connection Points

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

Java Message Service – a connection to a JMS Server or JMS Service.

A relational database, such as Oracle

A TCP/IP connection to a remote application using the HTTP or HTTPS protocol.

E-mail (uses standard SMTP for outbound and POP3 interfaces for inbound messages)

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

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

SeeBeyond Administration and Monitoring Tools

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.

e*Gate Schema Manager

This application 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.

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.

e*Gate Schema Designer

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.

e*Gate Alert Agents

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.

- 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.
- RIB Database Objects. These are Oracle Objects and tables as well as Message Family API stored procedures to support the Publishing and Subscribing e*Ways. They are part of the specific Retek application, such as RMS.
- 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 (Retek Merchandising System (RMS), Retek Warehouse Management System (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 (Integrated Store Operations (ISO) Store Inventory Management (SIM), for example), 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 SeeBeyond e*Gate Integrator EAI platform:

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

  These three manuals detail the options, configuration, and other reference material for creating Agents and other monitors for a deployed system.

Use the following resources to further understand the Retek RIB components implementation on the SeeBeyond e*Gate Integrator platform:

Retek RIB 11.1.x Technical Architecture Guide.

Chapter 2 – RIB Operation Overview

This section details the message flows for a simple message and for a message undergoing a routing or filtering operation.

Simple message flow

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.

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.

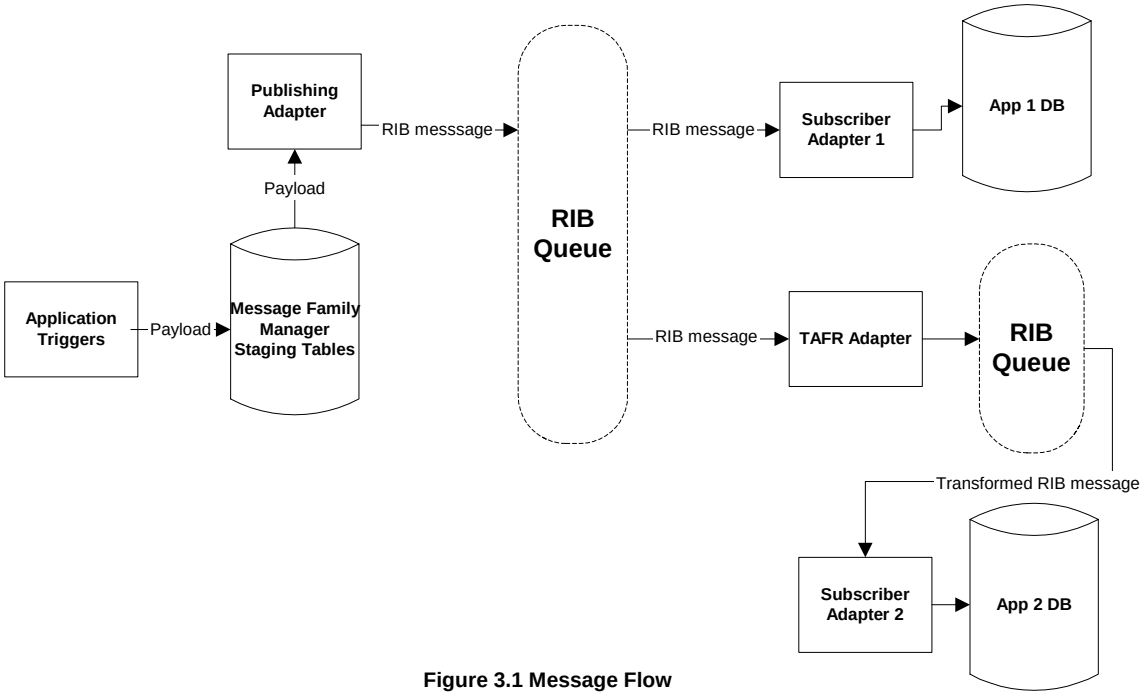


Figure 3.1 Message Flow

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. RWMS 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.

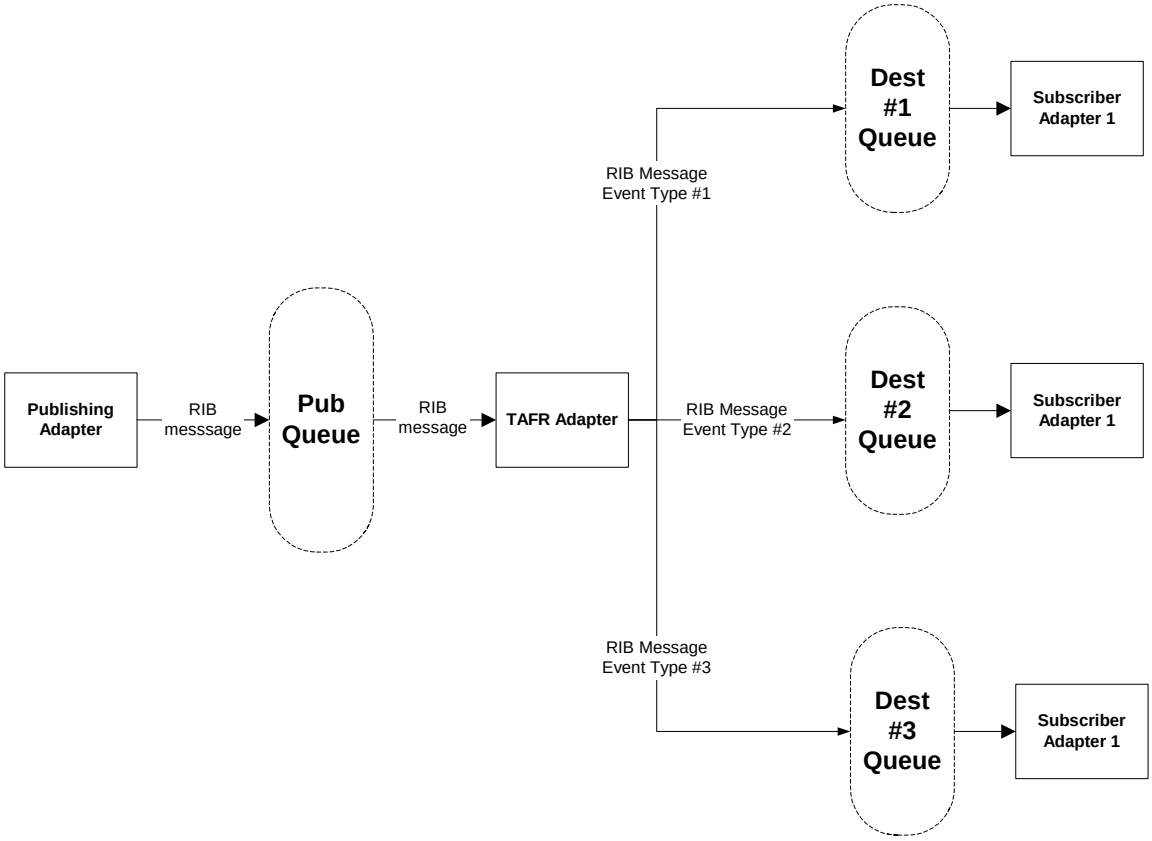


Figure 3.2 Routed Message Flow

Additional resources

Use the following resources to further understand the SeeBeyond e*Gate Integrator EAI platform:

- e*Gate Integrator System Administrator and Operations Guide
  Contains reference, troubleshooting and administrative information.

Use the following resources to further understand the Retek RIB components operation on the SeeBeyond e*Gate Integrator platform:

- Retek RIB 11.1.x Technical Architecture Guide.
- Retek RIB 11.1.x Integration Guide

Chapter 3 – RIB Configuration

The basic installation and configuration of the RIB is covered in the Retek RIB 10.3.x Installation Guide. This section provides additional details on the configuration options.

RIB Properties File

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 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 Files

Previous version of the RIB had a section in the rib.properties file that controlled these functions. In the current version these are controlled by the log4j.xml file. See the section on RIB logging. The properties file still contains these place markers:

```
##########################################
# Log e*Way times? Set in log4j.xml
##########################################
# Default logging level verbose? [Y or N]
# Set in log4j.xml
##########################################
# e*Way specific logging level verbose? [Y or N]. Set in log4j.xml
##########################################
# Path where RIB and Timings log files will be written.
# Set in log4j.xml
```

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.

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)`

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.

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.

RWMS 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).

FlowTrak specific entries

`prop.strm.fname` - `location of the FlowTrak properties file`

Other Application specific entries

All of these parameters are TAFR specific and are used in routing the messages to the correct application.

These boolean variables (True/False) are set to indicate which type of applications are active and which combination of to_loc/from_loc each application is designed to handle. If the message being sent contains both a “from location” and a “to location”, then the message could be sent to both the from and to location applications. However, not all systems can handle this information as a sender and receiver. This property was primarily developed for the SIM application. This application can handle the *Transfer* message when it is to be sent from a store Or if it is going to be shipped to their store. A lot of times the receiving location does not know or care about the transfer prior to receiving the *ASN Out* or *ASN In* message. In a pure Retek environment that contains both RWMS and SIM the setting next to the property below should be used. If message does not meet the conditions of the below properties, the ***message will be dropped***.

StoreSystemActive=True – This states that a store system is active and if a message is being routed it, should be routed to the store topic if the properties below also meet the conditions.

StoreSystemRouteToToLoc=True – This states that if the store is the to_loc of a message (such as a transfer) then the message should be routed to the store application topic.

StoreSystemRouteToFromLoc=True - This states that if the store is the from_loc of a message (such as a transfer) then the message should be routed to the store application topic.

WarehouseSystemActive=True – This states that a store system is active and if a message is being routed it, should be routed to the store topic if the properties below also meet the conditions.

WarehouseSystemRouteToToLoc=False - This states that if the warehouse is the to_loc of a message (such as a transfer) then the message should be routed to the warehouse application topic.

WarehouseSystemRouteToFromLoc=True - This states that if the warehouse is the from_loc of a message (such as a transfer) then the message should be routed to the warehouse application topic.

xternalSystemActive=False - This states that a n external system is active and if a message is being routed it, should be routed to the external topic if the properties below also meet the conditions.

ExternalSystemRouteToToLoc=False - This states that if a external ID is in the to_loc of a message (such as a transfer) then the message should be routed to the external application topic.

ExternalSystemRouteToFromLoc=False - This states that if the external ID is in the from_loc of a message (such as a transfer) then the message should be routed to the external application topic.

Hibernate.cfg.xml

Hibernate is an Open Source object relational mapping framework, and provides relational database independence in an object oriented fashion. It insulates an application from the details of accessing database tables and storing the data into Java objects. The RIB uses it for accessing the Error Hospital (checking dependencies, inserting new rows, updating status, etc).

In order for hibernate to validate the database connection, the hibernate.cfg.xml file located in the $EHOME/client/classes directory must have the following sections updated with valid database settings (url, database user, database password).

```
<property
name="hibernate.connection.url">jdbc:oracle:thin:@mspdev36:1521:devr
tk11</property>

<property name="hibernate.connection.username">seebeyond2</property>

<property name="hibernate.connection.password">rib104</property>
```

Component.xml

Previous versions of the RIB as deployed on the SeeBeyond platform used a collaboration rule specific to the Message Family for publishers and subscribing adaptors. In the 11.0.0 release, the number of different collaboration rules dramatically decreased. All publishers that call a GETNXT() stored procedure use the crGeneralPublisher collaboration rule and subscribers calling CONSUME() use either the crGeneralSubscriber or the crGeneralSubscriberRePublisher collaboration rule. This allows the same code to access CLOB or Oracle Object based Stored Procedures.

However, in order to correctly call the two API schemes, the stored procedure called and its parameters must be configured. The configuration file, component.xml, contains this configuration. This file is found in the $EHOME/client/classes directory. The tags found in this file are listed below:

component.xml Configuration File Entries	
Tag	**Description**
<rib-config>	Outer tag that contains the entire configuration. Contains one or more <eway> tags.
<eway name = “eWayName” >	Tag containing all entries for an e*Way. Attributes: **name** – the name of the e*Way Contains one or more <collaboration> tags.
<collaboration name = “collabName” >	Tag containing all entries for a specific collaboration within an e*Way. Attributes: **name** – the name of the collaboration Contains one or more <adaptorComponent> tags.

component.xml Configuration File Entries	
Tag	**Description**
<adaptorComponent>	Tag containing all entries for an adapter component found in the e*Way. The collaboration rule uses the adapter component to process the payload information. Contains one <class> and one <messageFamily> tag.
<class>	This tag contains the class name of the adaptor component.
<messageFamily name = “familyName”>	Tag containing Message Family specific information. Attributes: **name** -- the name of the Message Family. Contains one <storedProc> tag, one or more <messageType> tags, an optional <translatorClass> tag, an optional <rePubMessageType> tag, an optional <rePubOracleObjectType> tag.
<storedProc> signature>	Describes the stored procedure API. Contains one <signature> tag, zero or more <outParameter> tags. Contains the SQL to use to call the stored procedure.
<outParameter index=”somenumber”>	Describes an unusual parameter. Normally GETNXT() and CONSUME() have a standard set of parameters. When additional parameters are present – to return some set of values, this tag describes the position and type expected. Attributes: **index –** position of the parameter in the parameter list. Contains one <type> tag, may contain a <toJavaField> tag, may contain a <name> tag.
<type>	Describes the type of the output parameter. Contains a <value> tag and an optional <name> tag.
<value>	Contains the actual type value of the Oracle type to use. One of NUMERIC, VARCHAR, INTEGER, FLOAT, DATE, STRUCT, or ARRAY
<name>	Contains the name of the ARRAY or STRUCT type of the output parameter.
<translatorClass>	Used within a <messageFamily> tag for a publisher. Specifies the class to use to create XML from a RIB Object output from a stored procedure.
<rePubMessageType>	Used for subscribers that may also publish messages because the called stored procedure returns an output RIB Object. This tag contains the Message Type of the published message. Note: the published Message Family is the same on the published message as was found on the <MessageFamily> tag.
<rePubOracleObjectType>	Used for subscribers that may also publish messages because the called stored procedure returns an output RIB Object. Contains the name of the ARRAY or STRUCT type of the output parameter that is published.

component.xml Configuration File Entries	
Tag	**Description**
<messageType name="Message Type" >	Describes a Message Type / RIB Object relationship. Attributes: **name --** name of the message type. Contains one <oracleObject> tag.
<oracleObject>	Denotes the name of the Oracle Object used to contain payload information. Contains the name of the Oracle Object type defined in the database.
<toJavaField>	Used within an <outParameter> tag for CLOB publishers, this tag defines what <ribMessage> sub-tag that should contain the information of the output stored procedure parameter. If used in the <ID> tag to specify the Business Object ID of the message, the value is "ID", If used for a <RoutingInfo> tag, the value is of the form "ROUTING_INFO.<NAME>", where name is <name> tag found in the <routinginfo> tag. The value contained in the output parameter is placed into the <value> tag of the <routingInfo> record.

The XML below is a sample component.xml file containing two entries. One entry is for the Banner Message Family publisher (a CLOB API) and one is for a COReturn Message Family subscriber (a RIB Object API). Most component.xml files will contain many more entries.

```
<rib-config>
<eway name="ewBannerFromRMS">
      <collaboration name="colBannerFromRMS">
         <adaptorComponent>

<class>com.retek.rib.collab.general.CLOBPublisherComponentImpl</clas
s>
            <messageFamily name="Banner">
               <storedProc>
                  <signature>{call
RMSMFM_BANNER.GETNXT(?,?,?,?,?,?)}</signature>
                  <outParameter index="5">
                     <type><value>NUMERIC</value></type>
<toJavaField>ID</toJavaField>
                  </outParameter>
                  <outParameter index="6">
                     <type><value>NUMERIC</value></type>
                  </outParameter>
```

```
               </storedProc>
            </messageFamily>
         </adaptorComponent>
      </collaboration>
   </eway>

<eway name="ewCOReturnToRMS">
      <collaboration name="colCOReturnToRMS">
         <adaptorComponent>

<class>com.retek.rib.collab.general.OracleObjectSubscriberComponentI
mpl</class>
            <messageFamily name="COReturn">
               <storedProc>
                  <signature>{call
RMSSUB_CUSTRETSALE.CONSUME(?,?,?,?)}</signature>
               </storedProc>
               <messageType name="CUSTRETSALECRE">

<oracleObject>RIB_CUSTRETSALEDESC_REC</oracleObject>
               </messageType>
            </messageFamily>
         </adaptorComponent>
      </collaboration>
   </eway>
</rib-config>
```

Note there are two output parameters from the call to RMSMFM_BANNER.GETNXT(). The 5th parameter populates the <ID> tag in the <ribMessage> published, while the 6th is silently discarded.

Retek Binding configuration files

The term "Retek Binding" is used to identify a message processing subsystem. This subsystem consists of a set of classes used to process or create RIB messages and typically executes using the J2EE platform. It is responsible for translating the RIB Message payload XML string into a "payload" object and to execute application specific code to process the payload. The configuration of the Retek Binding is found in a set of property files

Properties files

binding.properties – The Retek Binding will look within a package, "com/retek/binding/rib" that is found on the CLASSPATH environment variable. The purpose of this properties file is to create a key from the RIB Message Family and Rib Message Type and map this key to an XML formatted mapping file. The mapping file is specific to the DTD describing the format of the message payload.

An example of the RIB Message Family and RIB Message Type key might be, "ASNOUT.ASNOUTCRE" for example. For more information on the XML mapping files, see the "XML Files" section, below. A typical entry in the binding.properties file might be:

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

In this example, the Retek Binding would look for the "ASNOutDescMap.XML" file in the "com/retek/binding/rib/payload" package, which would have to be on the CLASSPATH environment variable.

castor.properties – The Retek Binding builds on the open source Castor java-binding framework. The purpose of which is to "bind" an XML document to a "payload" java object. The results of this binding can go in either direction – either from a populated java object being marshalling into an XML document, or from an XML document being unmarshalled into a populated java object. This framework provides tools for generating payload java objects. Payload objects javabean-like objects that hold XML data and provide marshalling and unmarshalling methods. The framework also generates XML mapping documents (see the "XML Files" section, below).

XML documents can be passed into the payload object's unmarshal(…) method, which returns a payload object, or the marshal() method may be called on a payload object, returning an XML document. The `castor.properties` file comes into play during runtime to control some aspects of the marshalling and unmarshalling operations. The file is not required, and in our case is only used to prevent validation of the incoming or outgoing XML document. This validation can prove costly in terms of performance, so by default it is turned off. So the following entry is the key reason that the Retek Binding includes the castor.properties file:

```
org.exolab.castor.parser.validation=false
```

The other entries in the file are accompanied by comments explaining the use of the associated entry. This file should be in the CLASSPATH of the application using the Retek Binding.

castorbuilder.properties – This file is used by the Castor framework during the generation of the java payload objects. A java payload object is a javabean-like object, with a number of attributes for holding data from an XML document. Payload objects can be generated from the Castor framework without the presence of this file, and the only reason it is included with the Retek Binding is that we want all payload objects to inherit from a common superclass. The entry for this property is:

```
org.exolab.castor.builder.superclass=com.retek.binding.rib.payload.Payload
```

The other entries in the file are accompanied by comments explaining the use of the associated entry. This file should be in the class path when doing the generation of the payload classes.

injector.properties – The purpose of this file is to match a Rib message family and Rib message type key to the fully qualified class name of an application class implementing the Retek Binding ApplicationMessageInjector interface. This interface is for subscribing APIs, and provides a way for them to consume a Rib message. A typical entry in this file might be:

```
DIFFS.DIFFCRE=
com.chelseasystems.cs.dataaccess.rib.subscriber.DifferentiatorCreateInjector
```

In this example, "DIFFS.DIFFCRE", is the Rib message family and Rib message type key of the subscribing API.

payload.properties – The purpose of this file is to match 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 for holding data from an XML document. Each of these payload objects in inherited from a common Retek Binding Payload superclass, in the "com.retek.binding.rib.payload" package. This file is used in both publishing and subscribing APIs, as both need a payload object. Publishers need a populated payload object from which to marshal an XML document. Subscribers need a payload object, into which to unmarshal an XML document. A typical entry in this file might be:

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

In this example, "ASNOUT.ASNOUTCRE", is the RIB message family and RIB message type key of the API.

publisher.properties – This file is used by the Retek Binding to allow users to disable publishing of messages. The only entry is for a property called, "ribMessagePublishEnabled", and takes a value of either "true" or "false". The file should be contained within the CLASSPATH environment variable in use. Other applications have used this file to include other application-specific information.

XML files

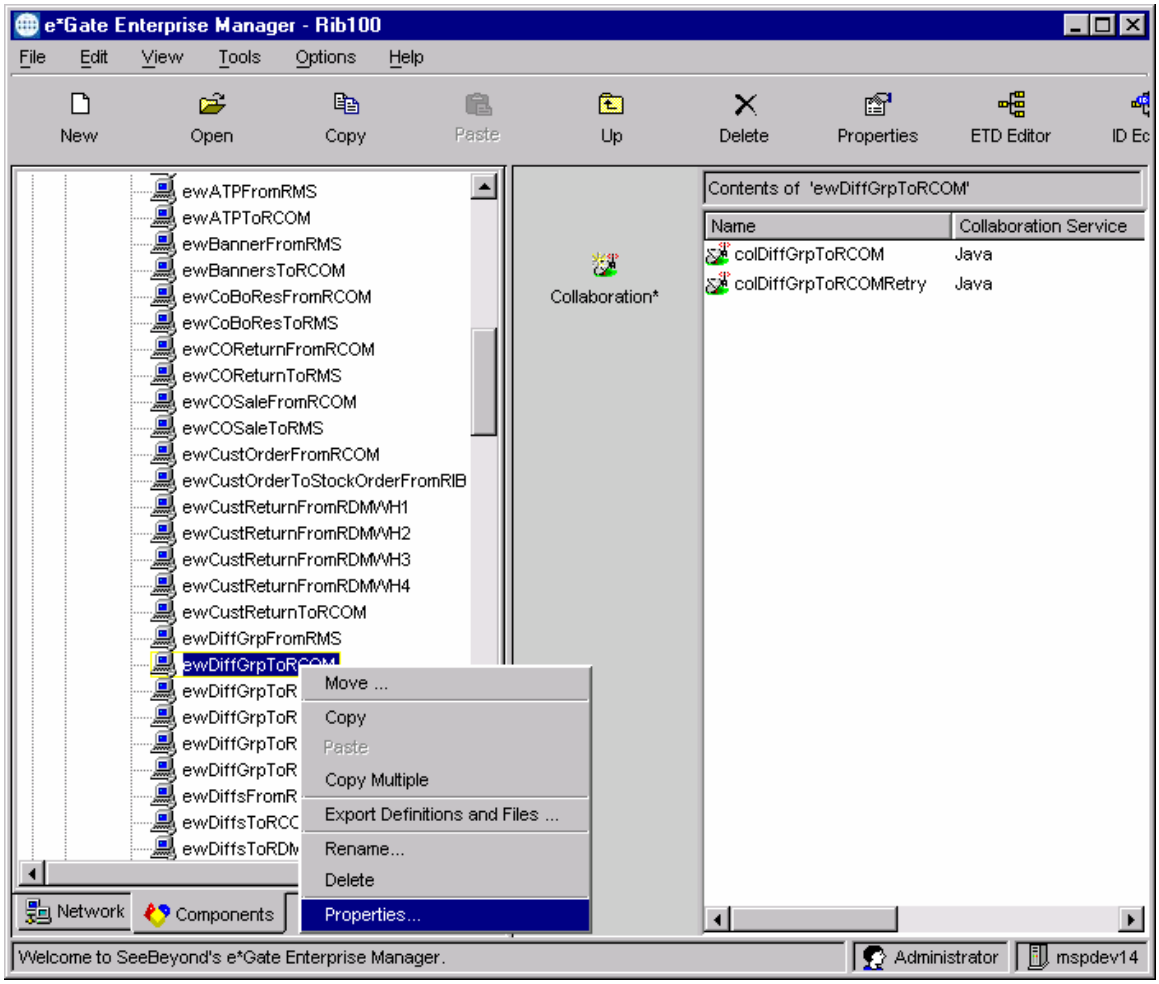
<PayloadObjectName>Map.xml – There is one of these files for each of the payload classes. These files map the element names in an XML schema document, to the attribute names in a java object. Because our payload objects are generated directly from an XML schema document, which in turn is generated from a DTD document, we theoretically should not need these files, although in testing the marshalling and unmarshalling of payload objects, this was not always the case. Here is an example of a "field" element from one of these files:

```
<field cst:name="item_id" cst:type="java.lang.String"
cst:required="true"><bind-xml name="item_id"
node="element"/></field>
```

These files are in the "`com.retek.binding.rib.payload`" package, which should be on the classpath of the application using the Retek Binding.

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.

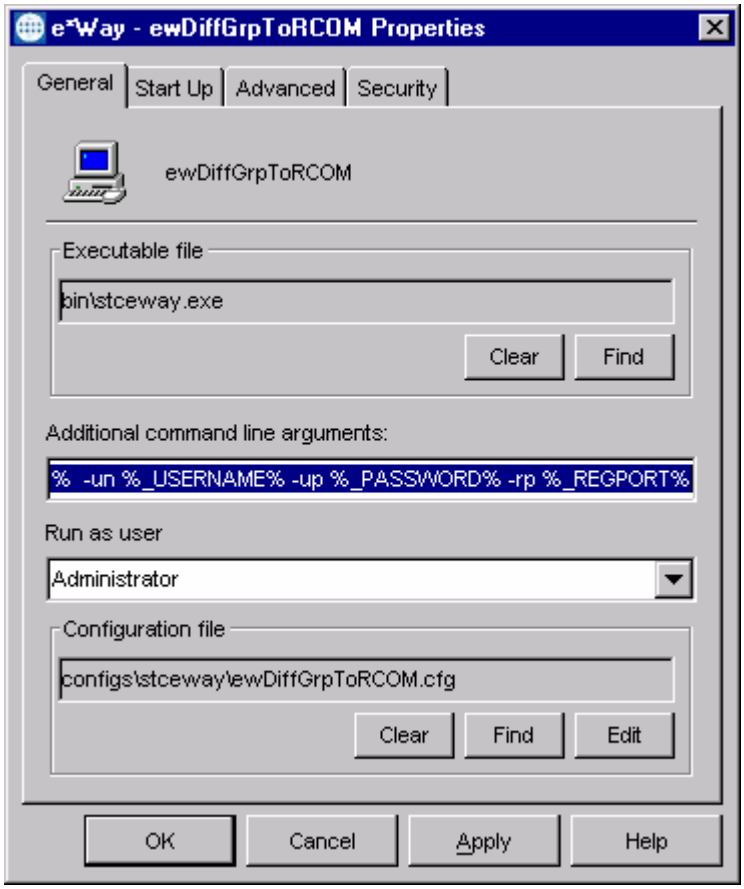


Right-click on e*Way in e*Gate Enterprise Manager

e*Way property and configuration files

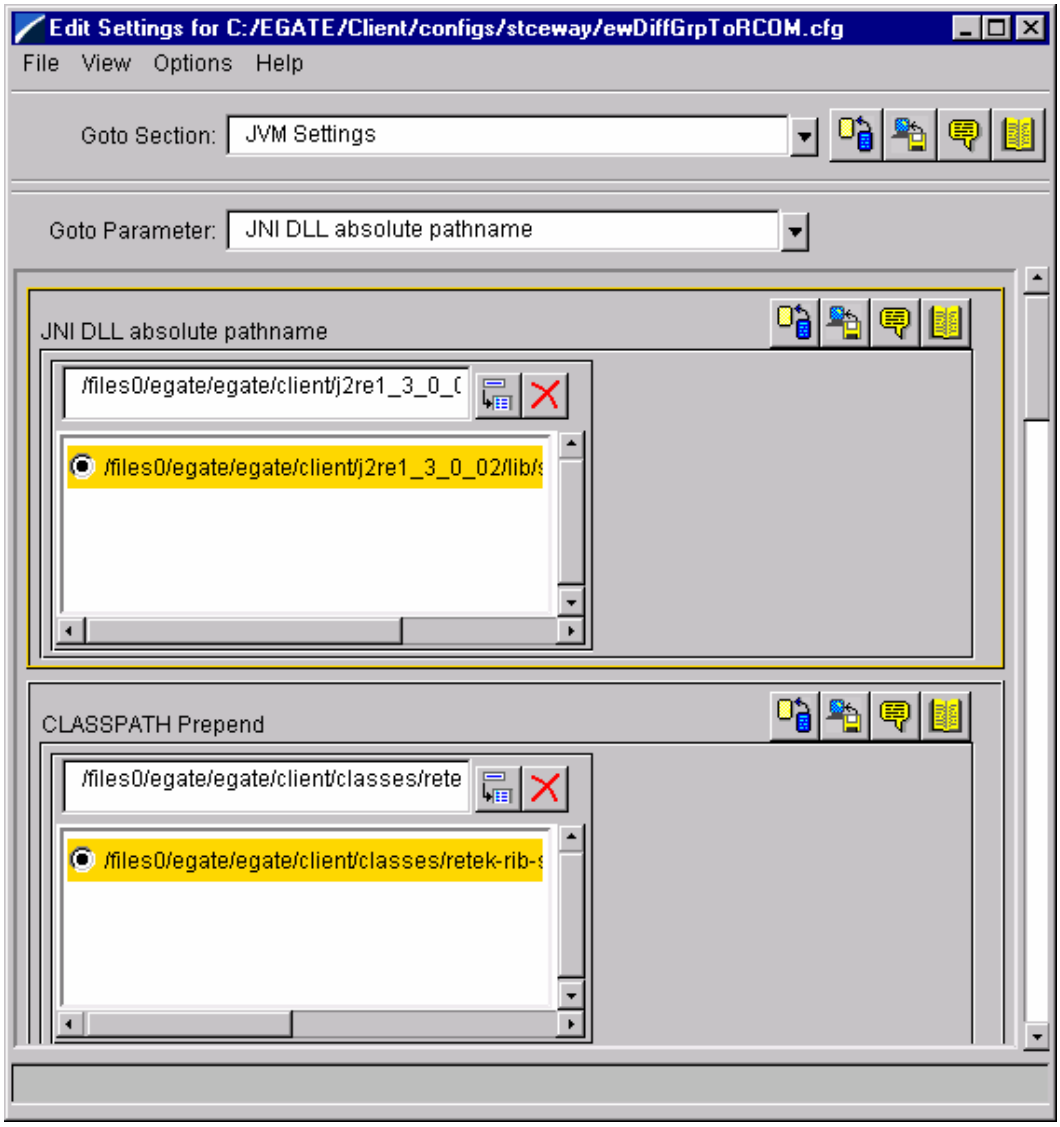
The following shows what is displayed when you right click to select an e*Way, to modify its properties.

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.so
  - (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

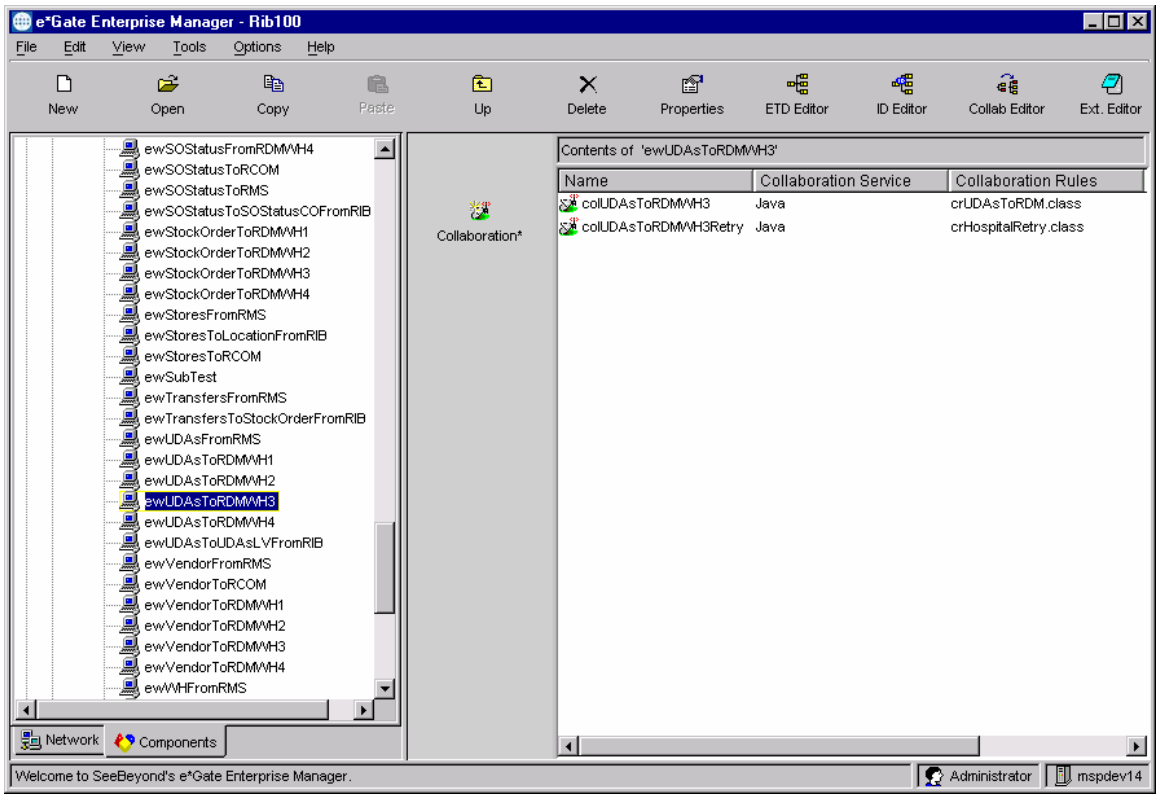
Note: The path separator is a semi-colon on the Windows system, and a colon on the Unix system.

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.

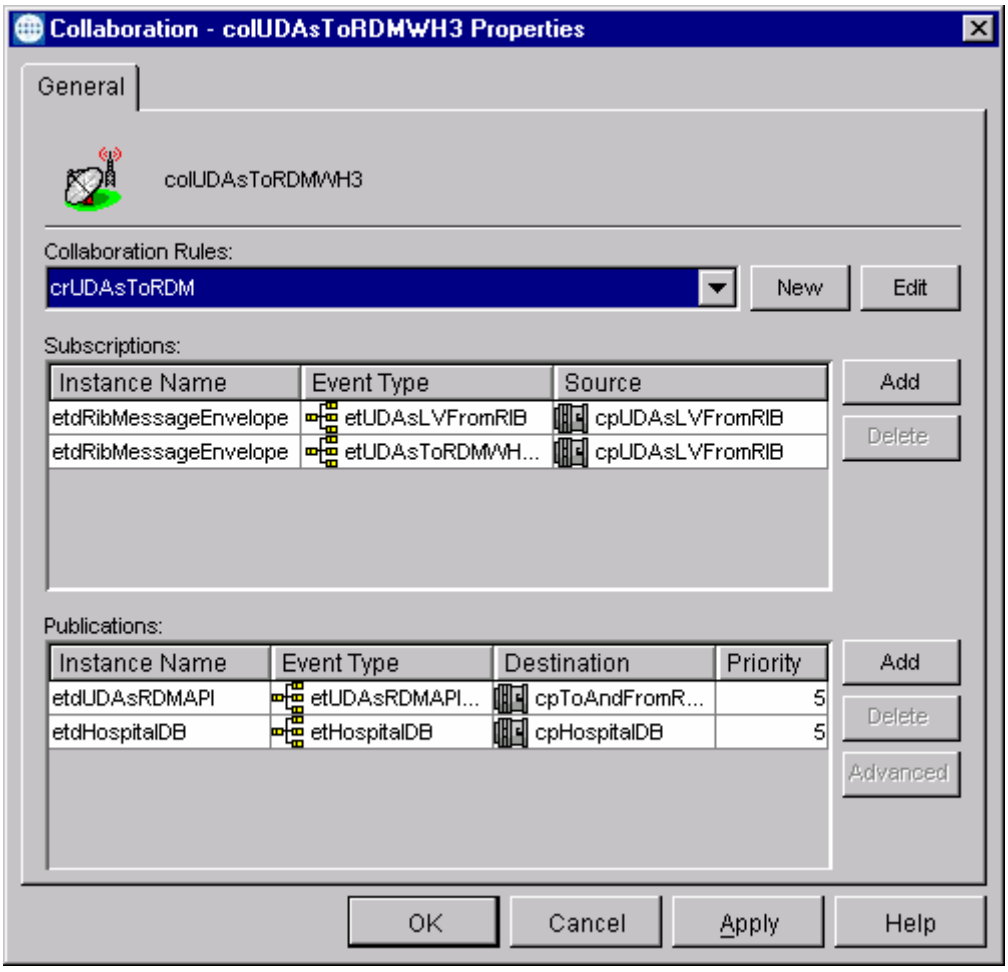
The figure below shows the main e*Gate Enterprise Manager for a RIB adapter.



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.

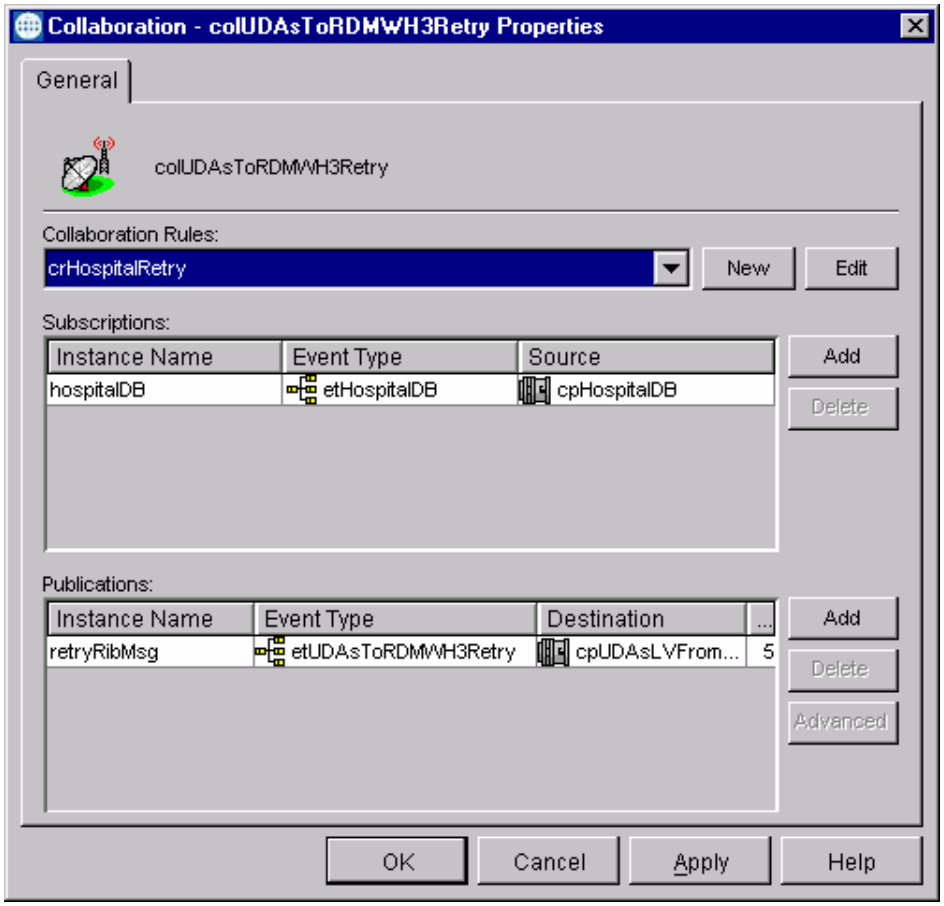
Note: This example uses a single JMS queue for all e*Ways in the EAI system. If a local queue were used, the connection point should be named something similar to cpUDAsLVFromRIBWH3.

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.

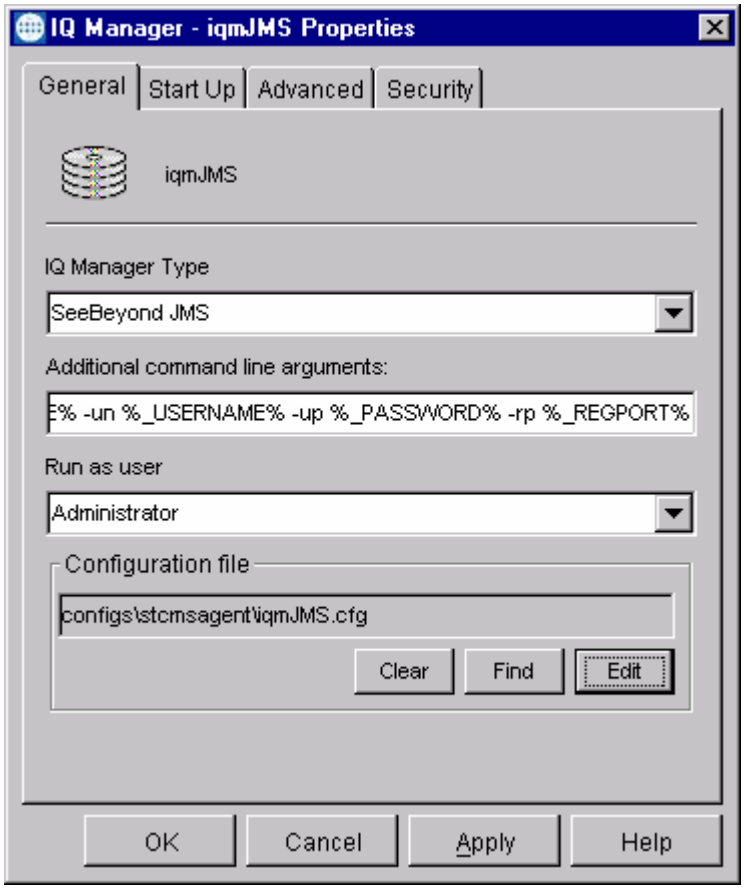
JMS IQ manager configuration

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.



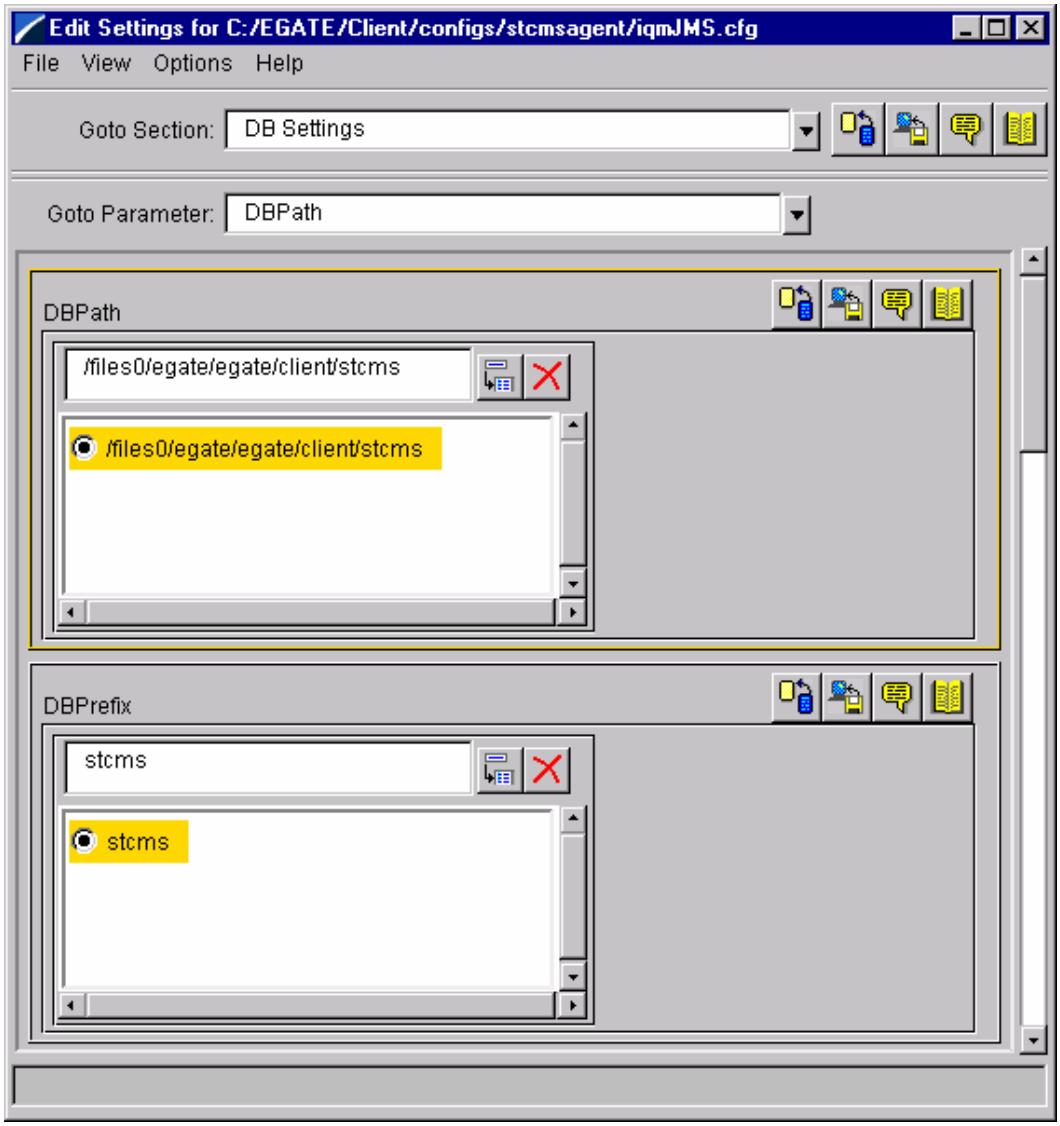
JMS IQ Manager Properties Window

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.

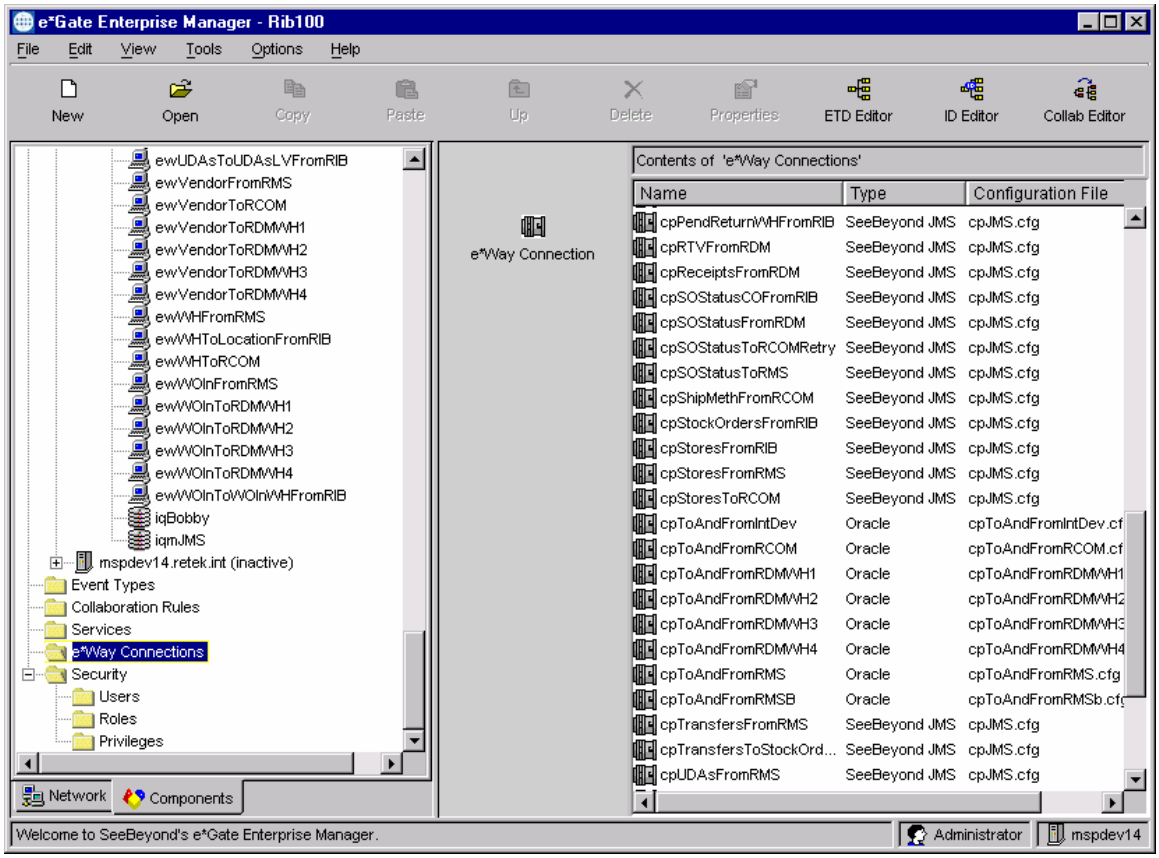
4 **Topic Settings:** This section sets the per-topic resource limits. In the RIB environment, a topic equates to an e*Gate Event Type which equates to a specific queue of messages supplying a set of subscribers.

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

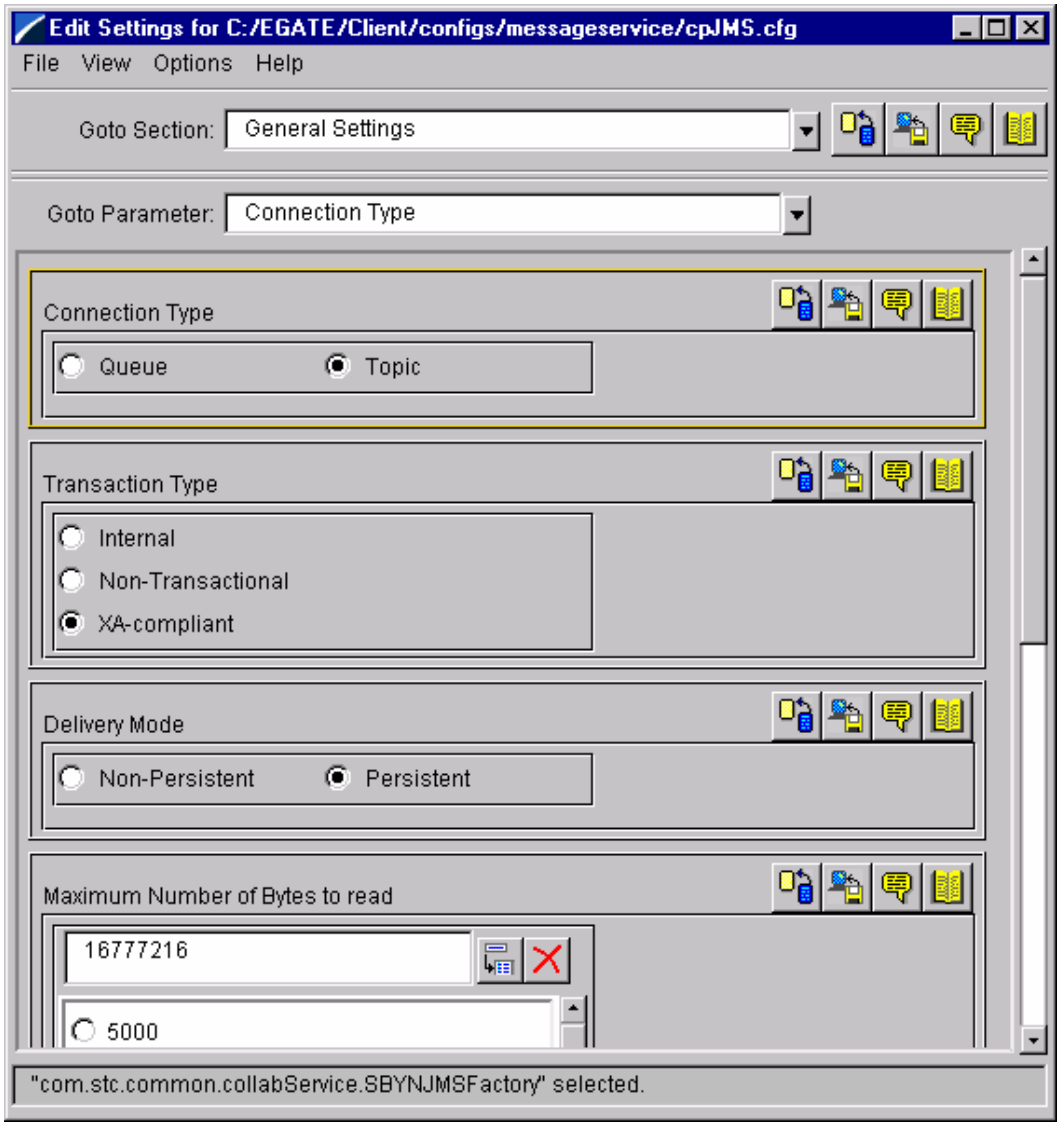
To create new connection points:

- 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
  ```

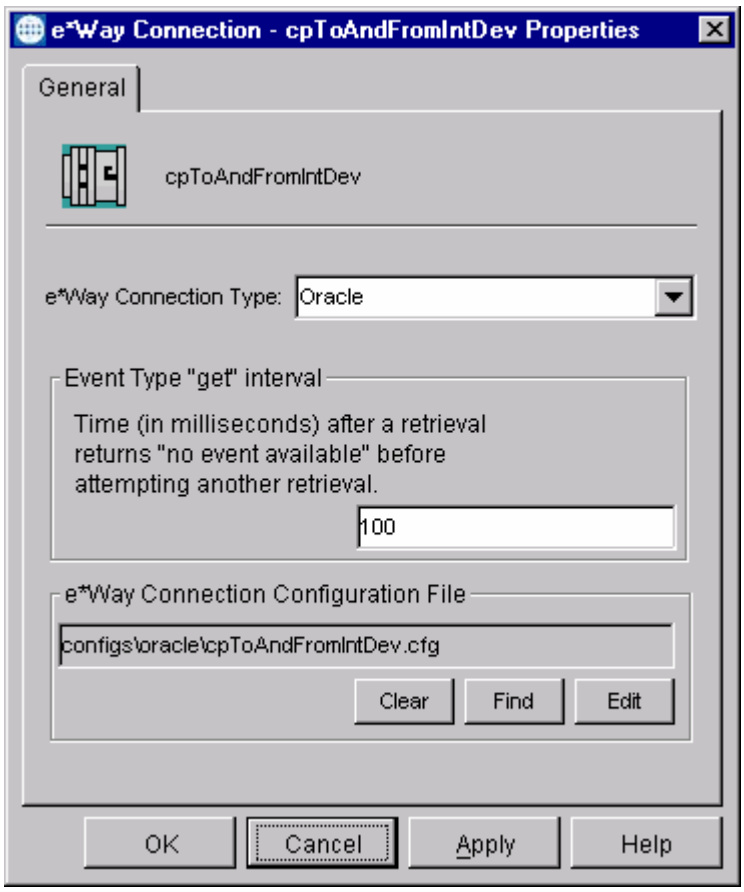
- Message Service: This section details JMS IQ Manager specific parameters for the JMS server.
- Server Name: Specifies the JMS IQ Manager name as seen in the e*Gate Enterprise Manager application.
- Host Name: Specifies the IP address or the host name from a Domain Name Server (DNS) that is running 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:

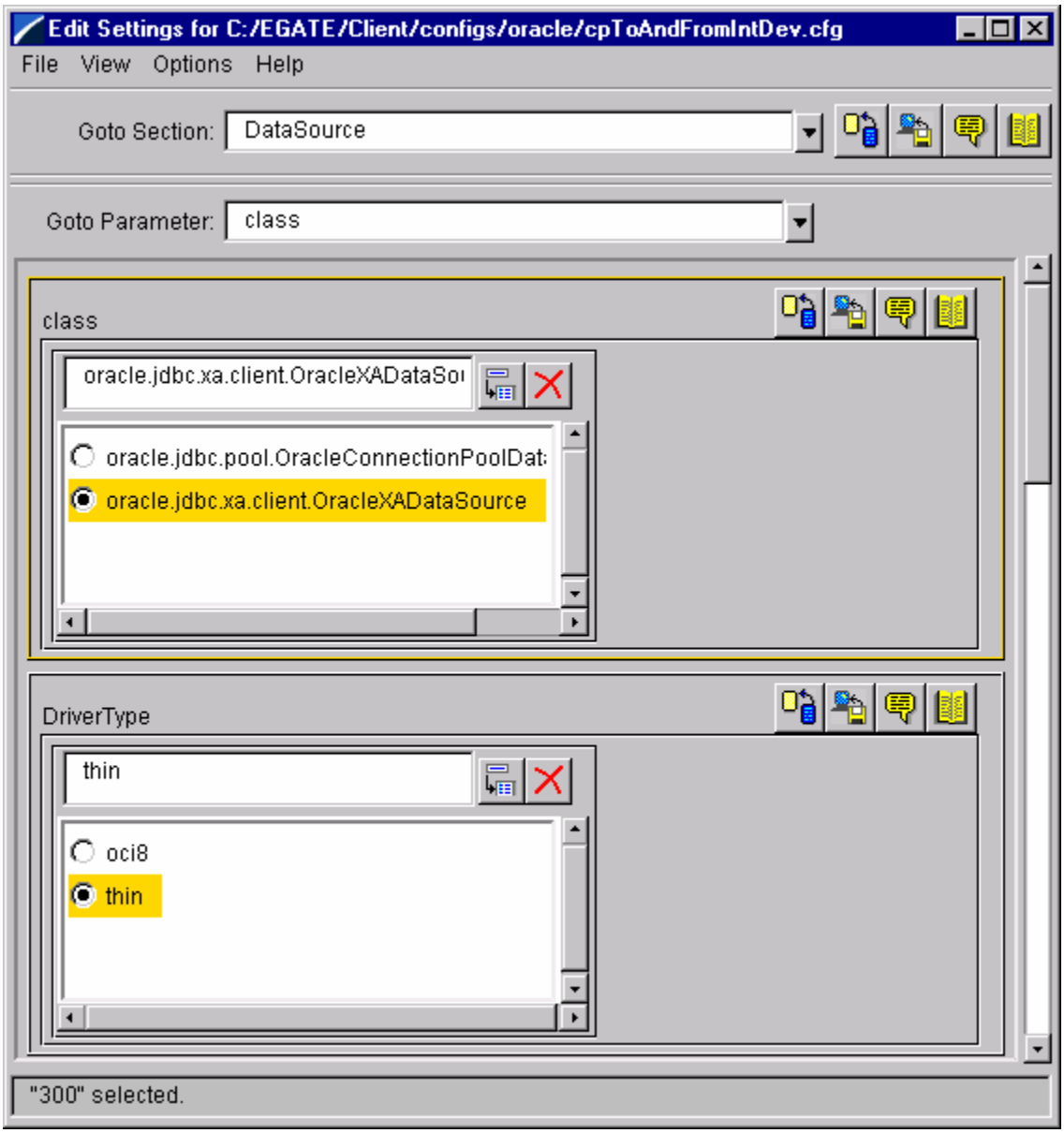


Oracle Connection Point Properties window

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

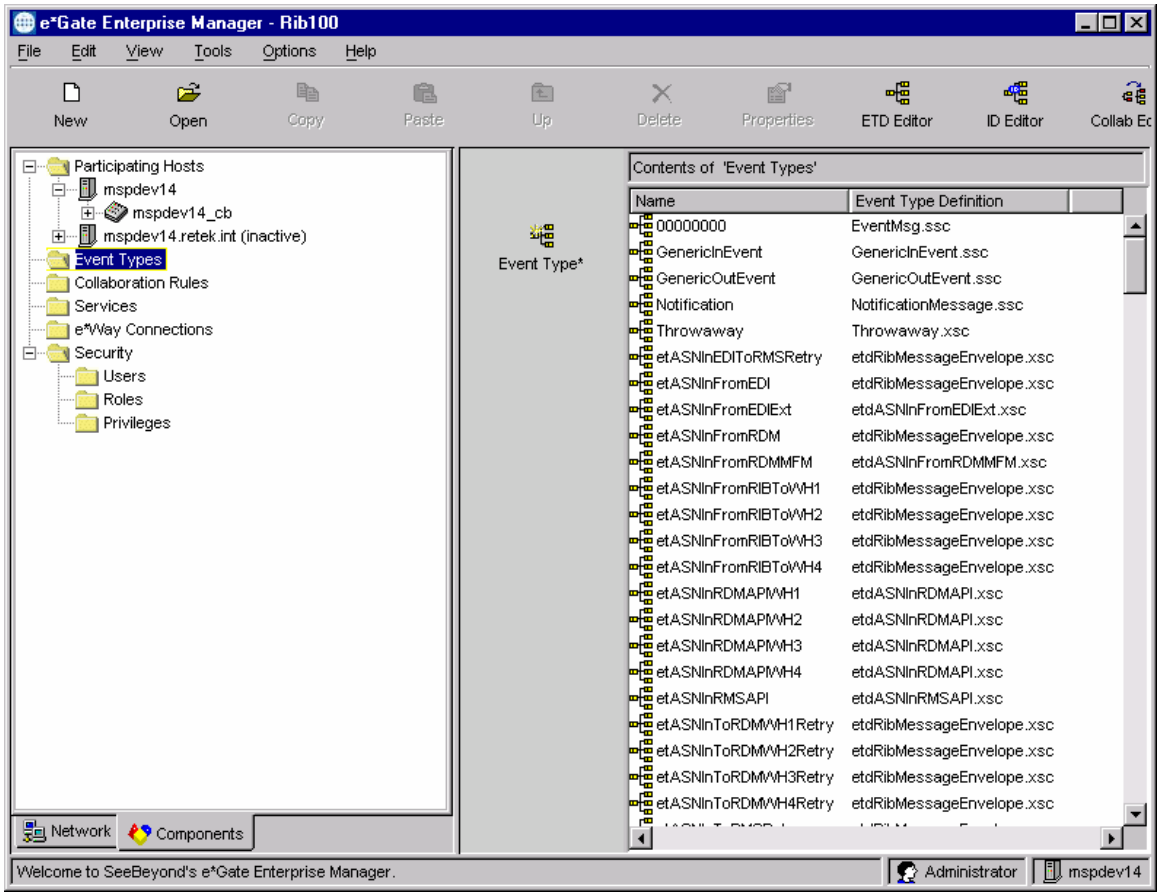
Two new event types will need to be created. The first is the new event type used by the TAFR component to route the message to the new destination. The second is used by the subscribing RIB adapter that interfaces with the application – the intended destination. These RIB e*Ways subscribe to two events, 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.

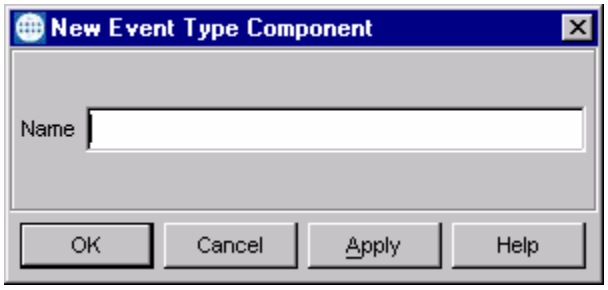
Once the name has been determined, the definition must be created. This is done via the e*Gate Enterprise Manager application. Clicking on the “Event Types” folder displays the following window:



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:



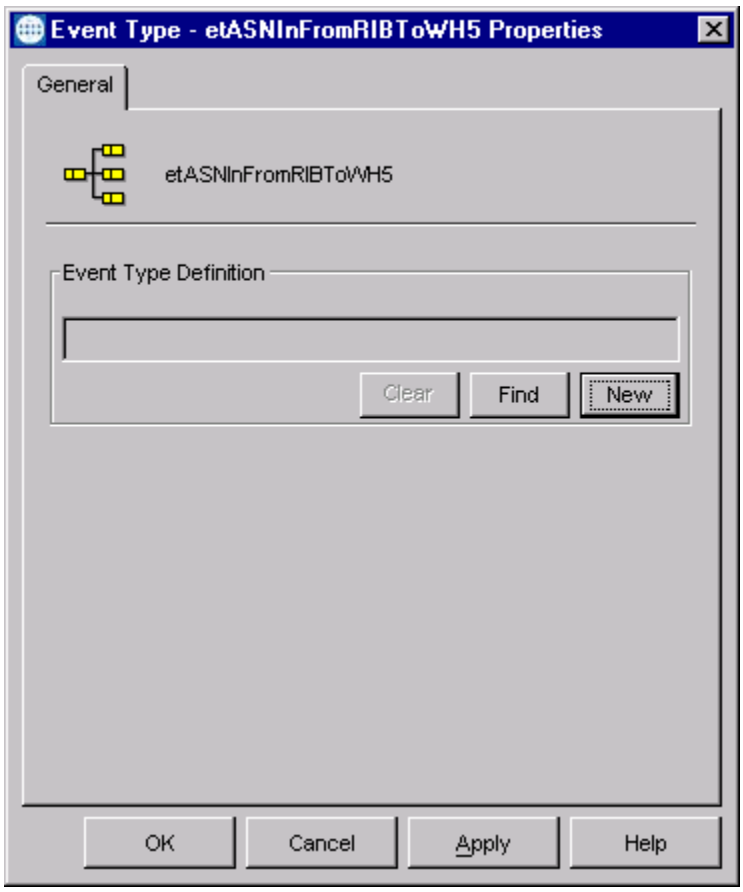
New Event Type window

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

2 Click **OK**.

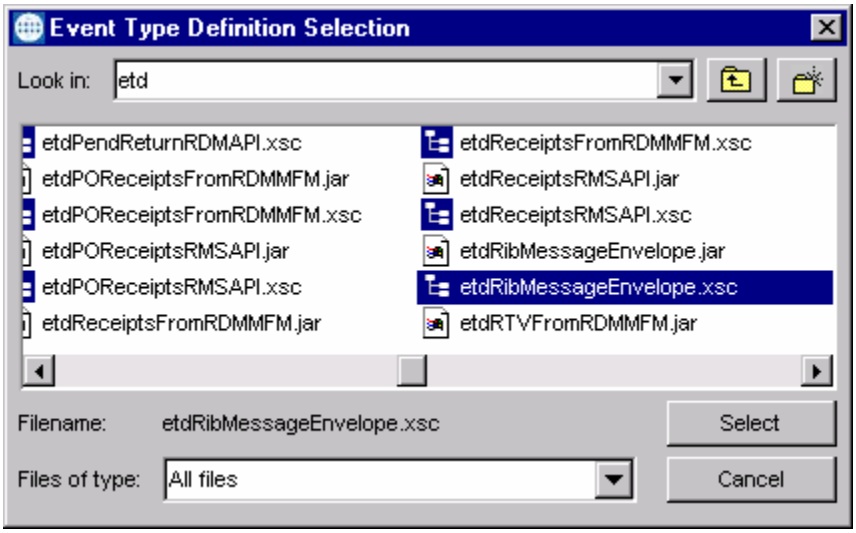
3 The new event type is displayed at the bottom of the list of event types.

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



Event type properties window

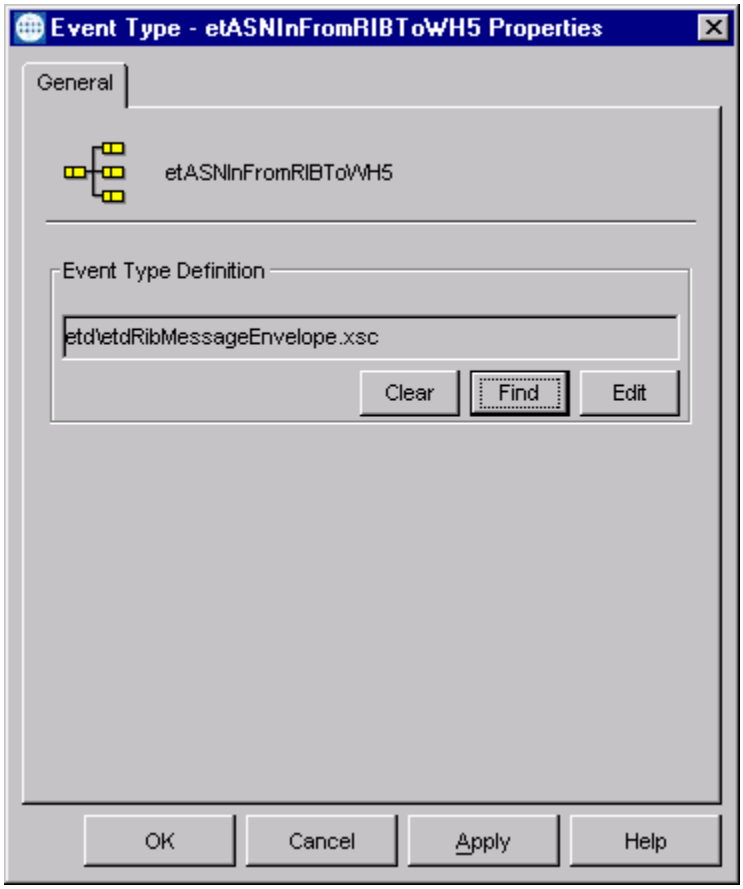
5 Click **Find**. This allows you to associate an existing message format (or Event Type Definition) with the new event type. (This may take a few seconds.) The Event Type Definition Selection window is displayed.



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.

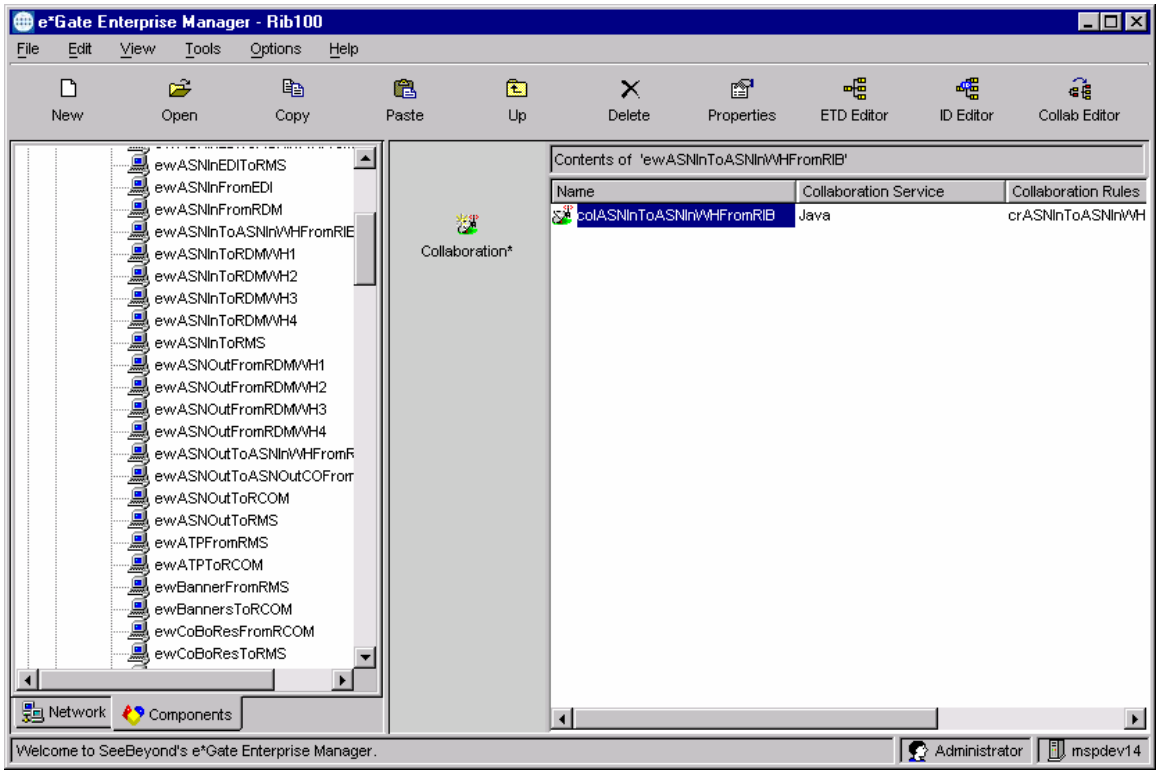
In the case of the examples above, the event type would be named etASNInFromRIBToWH5Retry.

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.

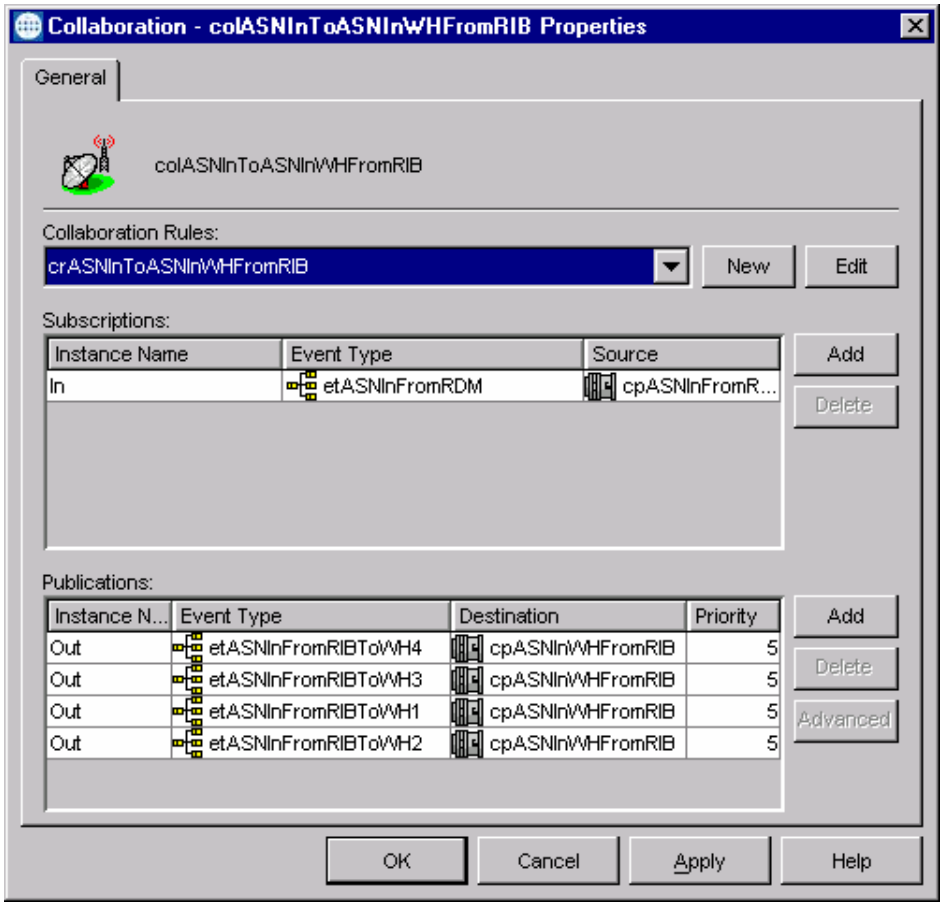


e*Gate Enterprise Manager with TAFR e*Way selected

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.

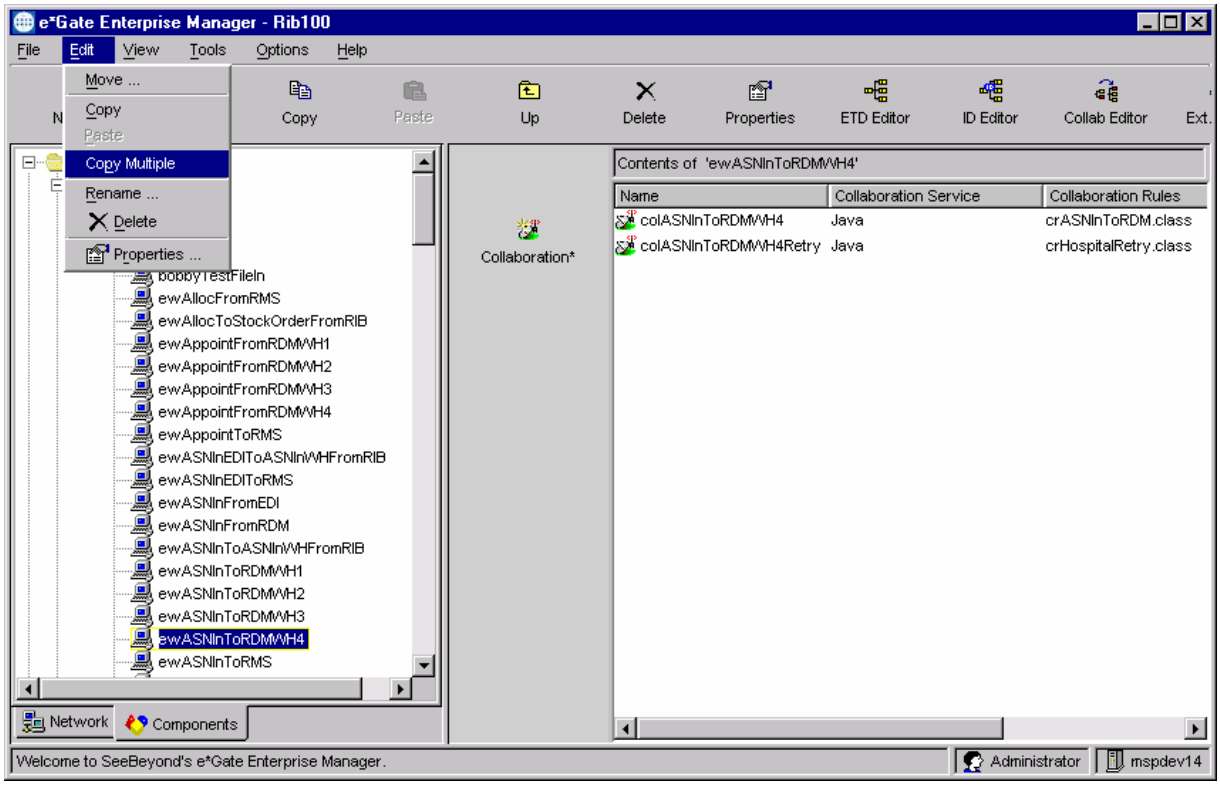
7 When the new event publication has been specified, click **OK** to save the information and update the e*Gate Registry with the new information.

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.

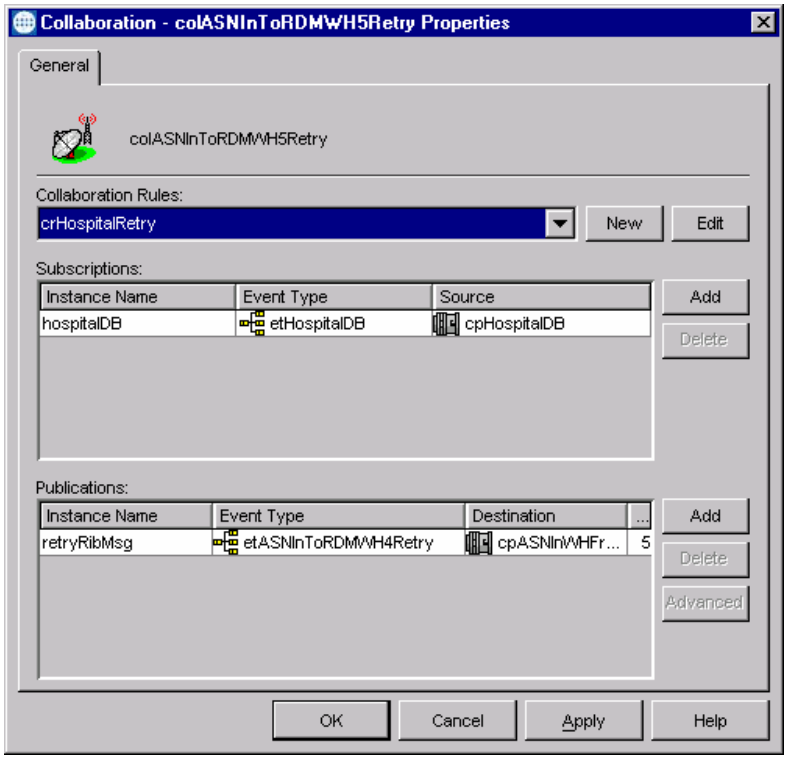


Copy Multiple edit option

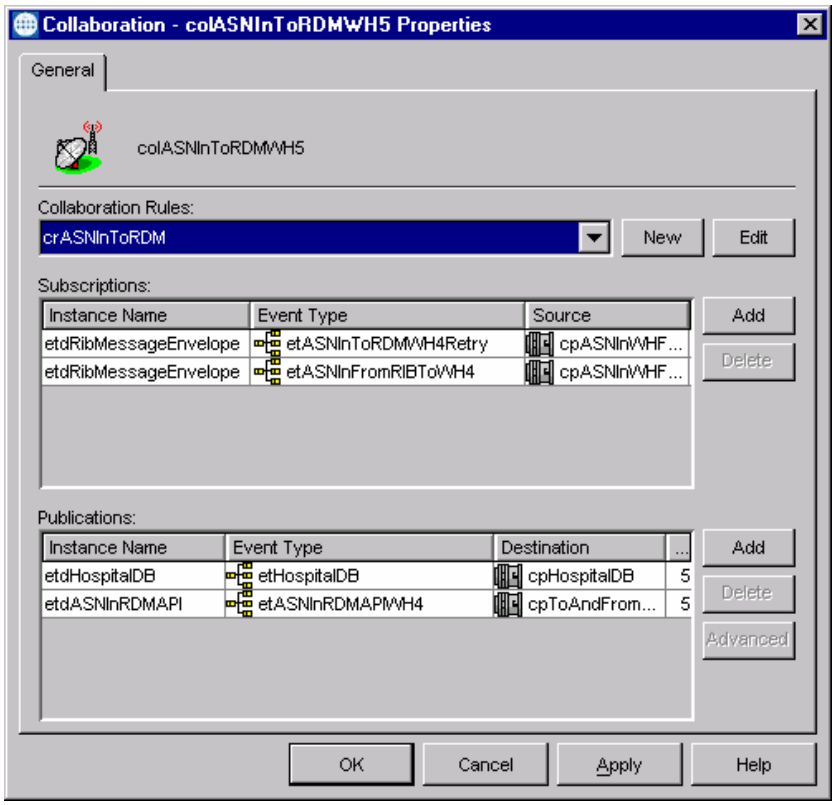
3 Rename the duplicate e*Ways to match the RIB's naming convention: For example, duplicating `ewASNInToRDMWH4` will result in `ewASNInToRDMWH4_0`. The RIB Naming convention renames the new e*Way to `ewASNInToRDMWH5`.

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.

Figures 7-21 and 7-22 show the Collaboration Property windows for a subscribing application. 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.

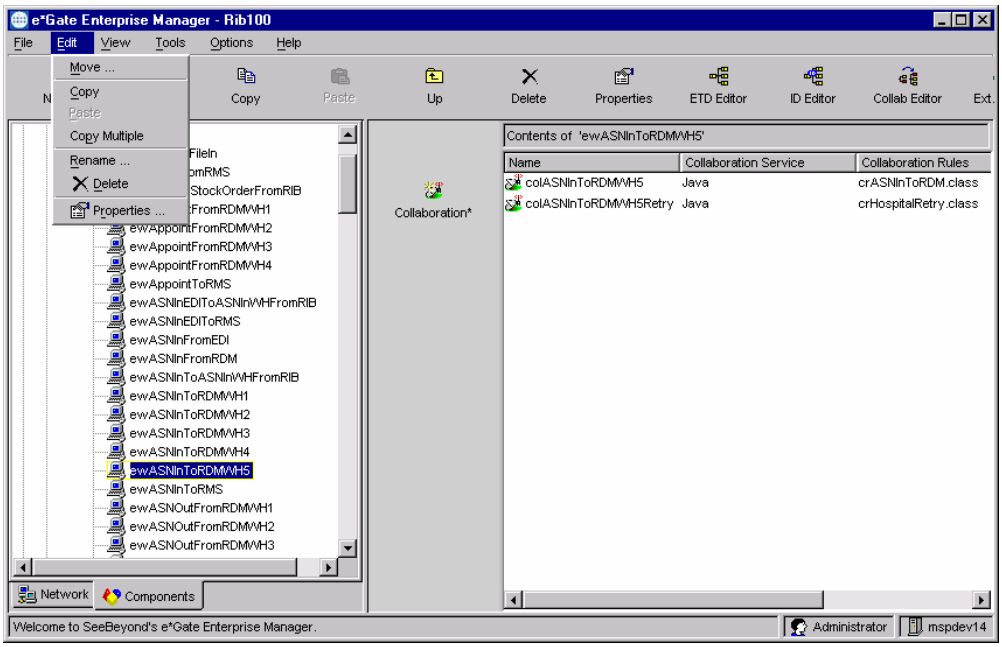
This connection point is associated with the etHospitalDB Event Type processing.

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…

Additional resources

Use the following resources to further understand the Retek RIB components configuration on the SeeBeyond e*Gate Integrator platform:

- Retek RIB 11.1.x Technical Architecture Guide.
- Retek RIB 11.1.x Installation Guide

Chapter 4 – 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.  **Note:** This script by default enables all alerts to be logged to the Notification Queue database. This database is mapped to the process space of the control broker. Unless there are appropriate methods to monitor and resolve e*Gate alert notifications, it is recommended that this script is modified and the –noe2n switch added to the command line.
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

 **Note:** These scripts 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. .

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 RIB1110

    -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, "//"

Example egate.txt:

// element name	Type	sequence
iqmJMS	JMS	1
ewHospitalRetryISO	HOSP	1
ewHospitalRetryRDM	HOSP	2
ewASNInFromEDI	PUB	1
ewASNInFromRDMWH1	PUB	2
ewASNInToRDMWH1	SUB	1
ewASNInToRMS	SUB	2
ewASNInEDIToASNInLocFromRIB	TAFR	1
ewASNOutToASNInLocFromRIB	TAFR	2

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.

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 “RIB1110”
- 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 RIB1110
–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.

Additional resources

Use the following resources to further understand the Retek RIB components configuration on the SeeBeyond e*Gate Integrator platform:

- e*Gate Integrator System Administrator and Operations Guide
  Contains reference, troubleshooting and administrative information.
- Retek RIB 11.1.x Technical Architecture Guide.
- Retek RIB 11.1.x Installation Guide

Chapter 5 – RIB Logging

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.

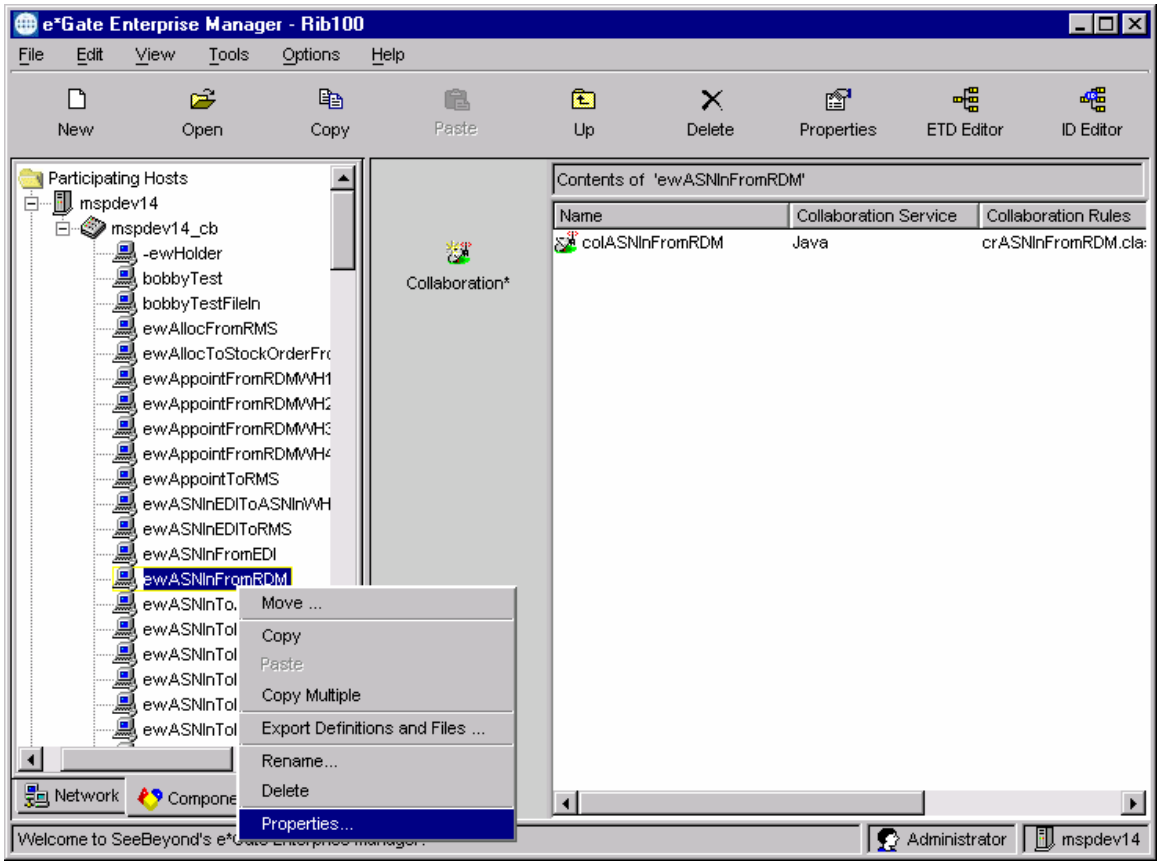
The location of the log files is the directory <EHOME>/client/logs, where <EHOME> is the installation directory for the SeeBeyond e*Gate EAI system. Each component has its own log file named <component>.log, where <component> is the name of the e*Way, control broker, or IQ Manager.

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:



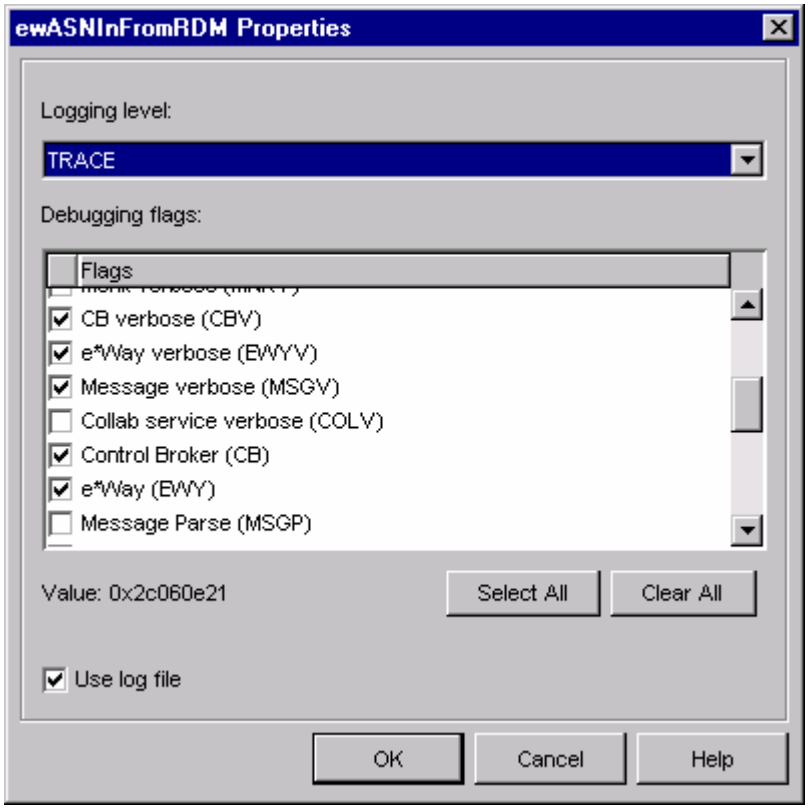
Selecting an e*Way from the e*Gate Enterprise Manager

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:

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

2 In the Logging File field, select TRACE.

3 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 Logging (RIBLOGS)

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 (see next section RIB Timing Logs).

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.

Note: The rib.properties <ewayName>.verbose entries no longer control the volume of logging into the RIB log files.

Log4j allows information to 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.

log4j xml configuration file

The Log4J Open Source software is used to control RIB logging of debugging, warning and Errors. This software requires the file “log4j.xml” to configure the file name, logging level, and type of file used. For SeeBeyond installations, this file is generated for each e*Way installed using the standard RIB installation scripts and placed into the $EHOME/client/classes directory. Sites that wish to use the same file for SeeBeyond logging and for RIB logging must edit this file.

Additional entries must be added for new e*Ways that use RIB Infrastructure.

A typical default entry for an e*Way looks like this:

```
<!--Eway Appenders-->
   <appender name="appender.retek"
class="org.apache.log4j.RollingFileAppender">
        <param name="File"
value="<EHOME>/RIBLOGS/rib_<e*WayName>.log"/>
        <param name="MaxFileSize" value="10MB"/>
        <param name="MaxBackupIndex" value="10"/>
        <layout class="org.apache.log4j.PatternLayout">
            <param name="ConversionPattern" value="%d{HH:mm:ss,SSS}
%c - %m%n"/>
        </layout>
  </appender>
```

```
<logger name="retek.<e*WayName>.<collaborationName>"
additivity="false">
         <appender-ref ref="appender.retek.
<e*WayName>.<collaborationName>"/>
    </logger>


Where
     <EHOME> is the installation directory of SeeBeyond. And
     <e*wayName> is the name of the e*Way
     <collaborationName> is the name of the collaboration found
within the e*way.
```

This entry specifies that the logging subsystem will create a file to store the RIB Log entries in the $EHOME/RIBLOGS directory. The file name will be rib_<e*WayName>.log. whenever this file is 10 megabytes in size, it will be renamed to rib_<e*WayName>.log.1 and a new rib_<e*WayName>.log will be created.. A maximum of 10 RIB Log files will be exist by default.

commons-logging.properties configuration file

In order to provide maximum flexibility for logging, the 11.1.0 RIB uses the Apache Commons Logging subsystem as the logging interface. If an installation wishes to use a logging subsystem other than log4j, then the file “commons-logging.properties” must be modified to specify the “Log Factory Implementation”. This implementation must implement the org.apache.commons.logging.LogFactory java interface.

RIB Timing Logs

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 to create a detailed performance report of the components of the RIB and its processes. The entries are recorded via writing to a timings log file. This information can be used to determine if the system is functioning correctly or if an application problem exists.

Typically, one timings log file is created per component (e*Way, EJB or other) which holds entries for that component during several life spans. These files are cumulative, meaning that they don’t get overwritten with every initialization of the component, but they append new entries to the current information already recorded. However these files do roll over after they reach a certain configurable size and backup files are created to preserve previous entries. The timings logs are written using the log4j logging mechanism; please refer to the previous section RIB Logging (RIBLOGS) for details about log4j. The timings log files follow the name convention timings_<ewName>.log.

Each entry in the timings log represents a timestamp of a particular event in the RIB component, listing the date/time information, name of the component, thread id and a distinct message for each event. The list of time stamped events comprehends such items as the start time and/or end time of the following actions:

- Overall publication, subscription, routing or transformation process
- Calls to store procedures (getnxt and consume)
- Actual publication and subscription of messages to and from the JMS server
- Calls to the Error Hospital to check for dependencies and insert messages
- Calls to other applications to process messages after subscription (injectors)
- Post-processing

Please refer to the Retek Integration Diagnostics and Monitoring Guide for a full discussion on how to perform post-processing of the RIB Timings Log, and more in-depth explanation of the timestamp events and performance analysis.

Additional resources

Use the following resources to further understand the Retek RIB components configuration on the SeeBeyond e*Gate Integrator platform:

- e*Gate Integrator System Administrator and Operations Guide
  Contains reference, troubleshooting and administrative information.
- Retek RIB 11.1.x Technical Architecture Guide.
- Retek RIB 11.1..x Installation Guide

Chapter 6 – Multi-Threading 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.

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.

How to Use it

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:

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.

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.

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.

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.

Additional resources

Use the following resources to further understand the Retek RIB components configuration on the SeeBeyond e*Gate Integrator platform:

- Retek RIB 11.1.x Technical Architecture Guide.

Chapter 7 – RIB and the 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 can uses the SeeBeyond JMS queue manager for integration between ISO, and other Retek modules such as RMS and RWMS. 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 “RIB Installation Guide: RIB component configuration: ISO Platform” for details regarding the configuration files.

ISO RIB LOGS

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

ISO 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 Schema Designer 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 – The RIB and 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.

RIB Logs

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 RIBfor*App* 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.

Error! Hyperlink reference not valid. port>/ribhospitalretry/ErrorHospitalRetryServlet

Chapter 9 – Troubleshooting (General)

This section lists a general approach to troubleshooting common RIB 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.

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'.

Messages re-tried from the Hospital that were 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.

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.

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.

Appendix A – RIB Hospital Administration Tool

Overview

The RIB Hospital Administration tool (RIHA) provides users with access to information contained in the application error hospital database. Users can view and modify this data and control the variables that make possible the feedback of messages into the system; they can retry, stop from retrying, delete or edit a RIB message. Each application typically uses its own set of Error Hospital database tables to store problematic messages; RIHA supports the configuration and access to multiple Error Hospital databases. Because of this, limiting the accessibility to this tool is imperative. RIHA supports the creation of user logins to guarantee that only designated users can execute this tool.

Installation and Configuration

RIHA provides a configuration utility to create and configure user logins and database connection information to access the multiple Error Hospital databases in a customer environment. Please see the Retek® Integration Bus™ 11.1.0 Installation Guide for instructions on how to execute this utility and configure RIHA for the first time.

Once RIHA has been installed, the riha-config utility can be executed again to include more user logins, add more database connections or modify existing ones. Start the utility by executing the riha-config.bat file on Windows or riha-config.sh script on Unix.

The utility will first prompt the user to enter a valid user id in order to access the configuration menu. Only users that were created during the installation steps will be able to modify the RIHA configuration.

```
Starting RIHA configuration utility...


Please login before continuing:

Enter User Id: jdoe

Enter password:  <password does not show>
```

Once the user has login, it is presented with the main configuration menu:

```
What would you like to do?

1) Add more users
2) Add/update database connections

([1], [2], [E]xit): 1
```

If the user chooses to add more user logins, it will be prompted to enter the new user information:

```
Create a new user login:

Enter User Id: janed

Enter User First Name: Jane

Enter User Last Name: Doe

Enter password:  <password does not show>

Verify password:  <password does not show>


User janed created.
Do you want to create another user? (y/n): n



What would you like to do?

1) Add more users
2) Add/update database connections

([1], [2], [E]xit): 2
```

The second option allows the user to either add a new database connection or update an existing one. A new menu is presented that displays the current database connections and an option to add more:

```
Select one connection to be re-configured, or create a new one.
Select [D]one when finished:

1) RMS - schema_owner@mspdev05:1521:DEV
2) Create new entry

[1], [2], [D]one): 1
```

In this example, the user chooses to update the existing database connection. The user is prompt for new values for each field or hit enter to accept the existing value displayed within parenthesis. At the end of this process, the database connection configuration menu is displayed again, reflecting the changes just made.

```
Enter new values or hit enter to accept existing value:


Enter value for the database host name (mspdev05): <current value
accepted>

Enter value for database port (1521): 1522

Enter value for database instance (DEV): <current value accepted>

Enter value for user name (schema_owner): new_owner

Enter value for password (Press Enter to keep current value):
<current value accepted>

Enter value for server hosting dtd files (mspdev05): <current
value accepted>

Enter value for server port hosting dtd files (8080): <current
value accepted>



Select one connection to be re-configured, or create a new one.
Select [D]one when finished:

1) RMS - new_owner@mspdev05:1522:DEV
2) Create new entry

[1], [2], [D]one): 2
```

The next example shows the steps followed to create a new database connection. The database connection create menu is displayed and the user chooses the application that hosts the Error Hospital database. Once configured, the new entry is displayed in the database connection configuration menu:

```
Please choose a product for configuring database information:

1) AIP  - Retek Advanced Inventory Planning
2) ISO  - Retek Integrated Store Operations
3) RCOM - Retek Customer Order Management
4) RMS  - Retek Merchandising System
5) RPM  - Retek Price Management
6) RWMS - Retek Warehouse Management System

([1], [2], [3], [4], [5], [6], [D]one): 2


Enter value for the database host name (e.g.: mspdev05.retek.int):
mspdev06

Enter value for database port (e.g.: 1521): 1526

Enter value for database instance (e.g.: DEV): ISODB

Enter value for user name: dbuser

Enter value for password:  <password does not show>

Verify password:  <password does not show>

Enter value for server hosting dtd files (e.g.: mspdev05.retek.int):
mspdev05

Enter value for server port hosting dtd files (e.g.: 8080): 8080




Select one connection to be re-configured, or create a new one.
Select [D]one when finished:

1) ISO - dbuser@mspdev06:1526:ISODB
2) RMS - new_owner@mspdev05:1522:DEV
3) Create new entry

[1], [2], [3], [D]one): d
```

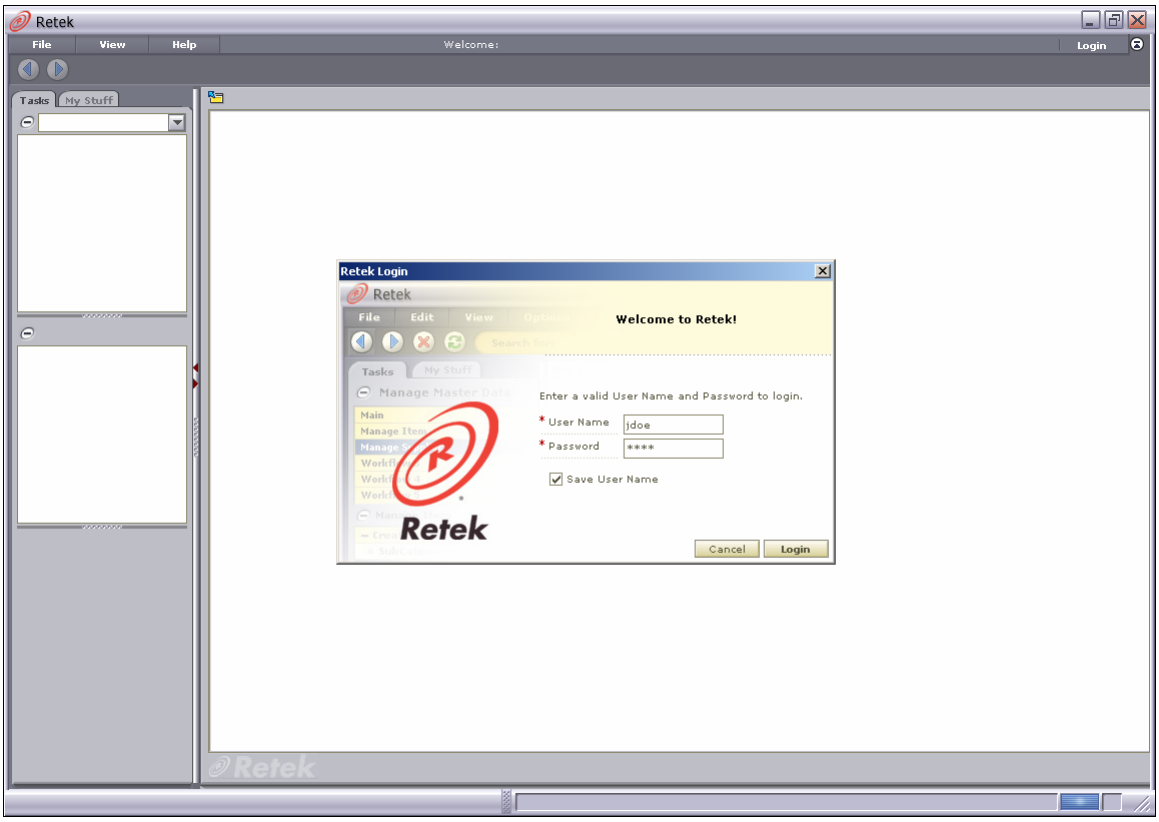
Selecting [D]one returns the user to the main menu, from which the user can choose to exit the application.

```
What would you like to do?

1) Add more users
2) Add/update database connections

([1], [2], [E]xit): e


RIHA configuration completed.
```

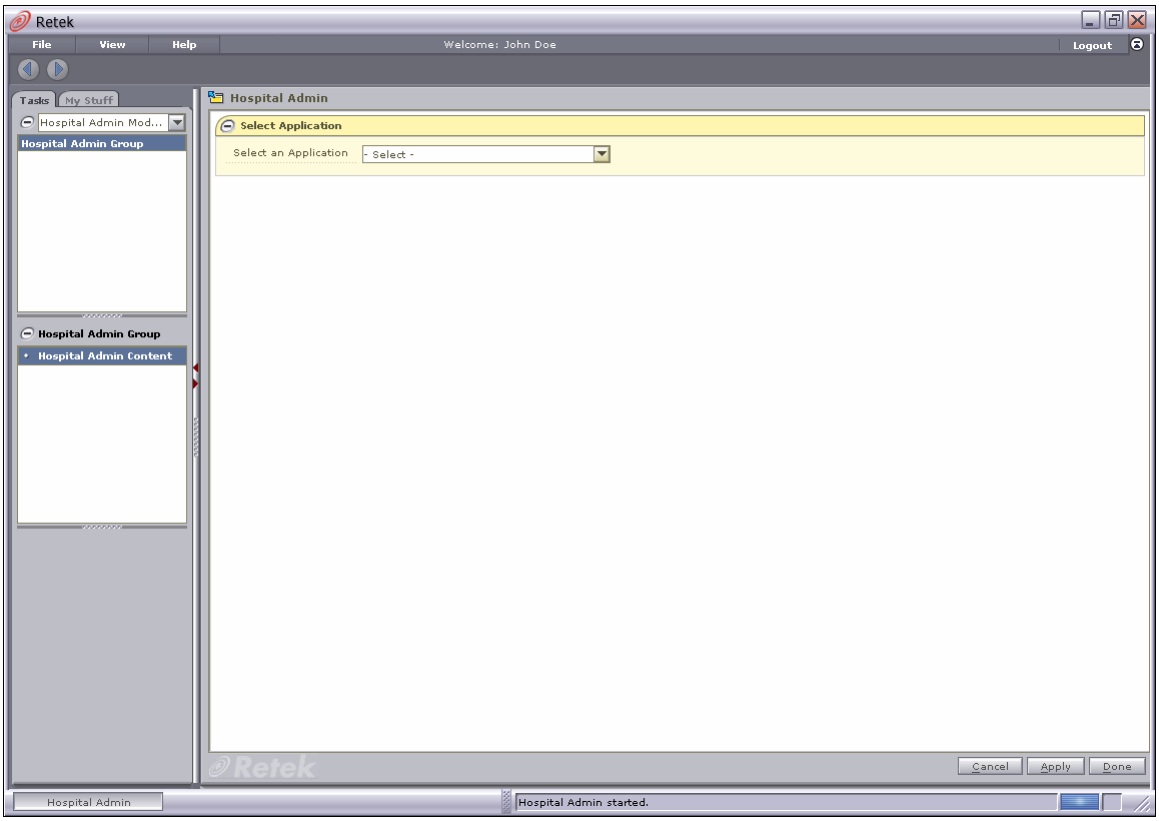
Accessing the RIB Hospital Administration tool

To start RIHA, change directories to RIHA1110\bin and execute the riha.bat file on Windows or riha.sh script on Unix.

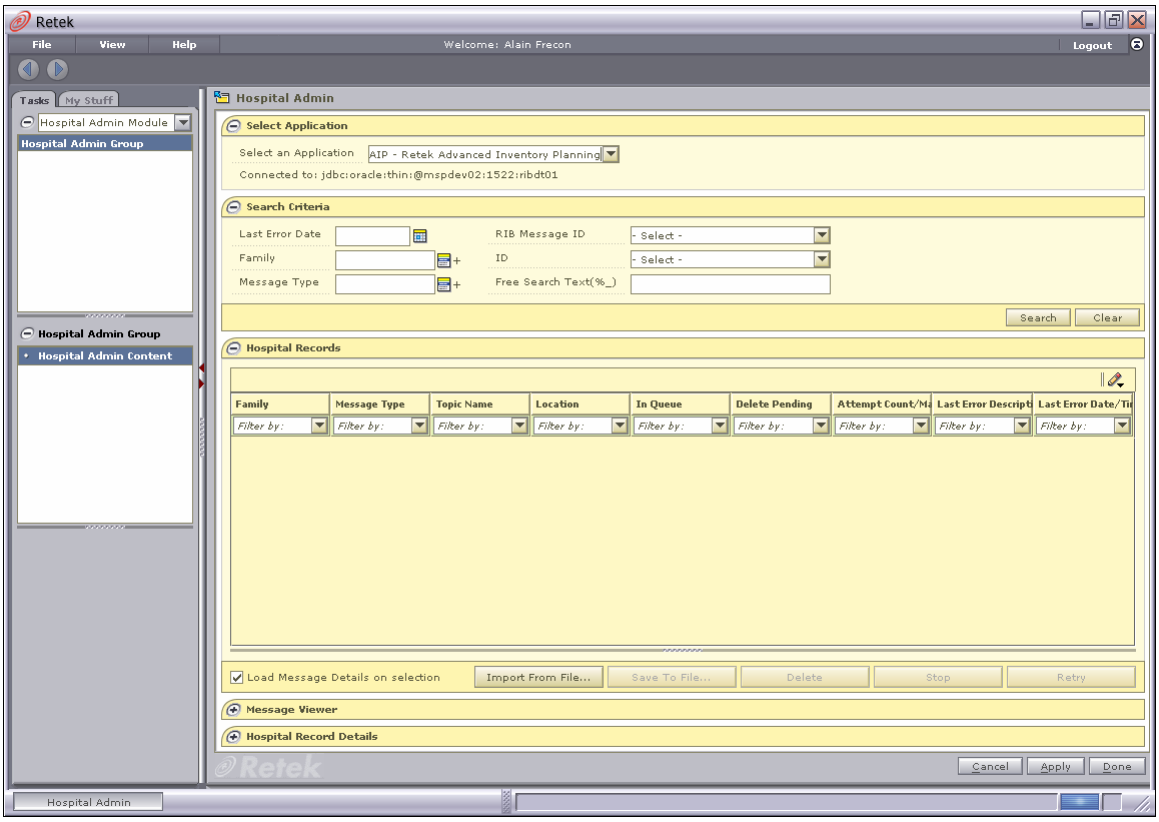
When the GUI starts up, a login window is displayed. Enter the user name and password of one of the logins created during installation or configuration. Hit the login button to continue.



After a successful login, the task pad in the left-hand side of the screen is populated with the Hospital Admin Module option in the upper portion of the pad and the Hospital Admin Content option in the lower portion. If the lower portion of the task pad does not show the Hospital Admin Content, select the Hospital Admin Module in the upper panel to load it. Then click on the Hospital Admin Content to load the RIB Hospital Administration content in the main workspace.



Navigate to the Select Application panel and click on the drop-down menu to select the application for which to display records in the Error Hospital.



Upon successful connection to the database, other panels are loaded that allows the user to search records, view their details and perform other activities related to the maintenance of these failed items. Please refer to the RIB Hospital Administration User Guide for a complete discussion of each panel and how to navigate the tool.