Retek® 10 Integration Bus

Operations Guide

Retek

Corporate Headquarters:

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

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

European Headquarters:

Retek
110 Wigmore Street
London
W1U 3RW
United Kingdom

Switchboard:
+44 (0)20 7563 4600

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

Customer Support

Customer Support hours:

8AM to 5PM Central Standard Time (GMT-6), Monday through Friday, excluding Retek company holidays (in 2002: Jan. 1, May 27, July 4, July 5, Sept. 2, Nov. 28, Nov. 29, and Dec. 25).

Customer Support emergency hours:

24 hours a day, 7 days a week.

Contact Method	Contact Information
Phone	US & Canada: 1-800-61-RETEK (1-800-617-3835) World: +1 612-587-5000
Fax	(+1) 612-587-5100
E-mail	support@retek.com
Internet	www.retek.com/support Retek's secure client Web site to update and view issues
Mail	Retek Customer Support Retek on the Mall 950 Nicollet Mall Minneapolis, MN 55403

When contacting Customer Support, please provide:

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

Contents

Chapter 1 – Introduction

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.

Chapter 2 – RIB Component Overview

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

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.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

The typical lifecycle of a message is as follows:

First, the publishing adapter creates the message. The event that triggers the message creation may be a polling operation on the database, the presence of a file, or merely that a certain time interval has been reached. Each message is created in the context of collaboration, and part of the collaboration's configuration specifies where to publish the created message. The message is sent to a "queue" that then writes the message to stable storage.

The message is now available to its subscribers. Subscription is based on the publishing collaboration / event type combination. Each subscriber will contact the queue and retrieve the next message available. Separate threads in the subscriber are used to retrieve messages on a per event type basis. The specific message retrieved from the queue depends on its location within the queue. As part of the retrieval processes, the Error Hospital software updates the state of the message to reflect that one of the subscribers is now processing it.

Once a subscriber gets the message, it is free to process it according to its own rules. In the case of a transformer adapter, the subscribing collaboration can open the message, modify its contents, and then publish the modified message to a new queue. If the new message is of a different type than the original, the new message can be published to the original queue. There may be new subscribers to the modified message, and the scenario repeated for each of these subscribers.

When each subscriber has finished processing a message, the queue updates the state of the message to reflect this. When all subscribers have finished with the message the message may be deleted immediately or be archived/journal led for a specific time before deletion. The archiving/journaling is specific to the type of the queue in use and the configuration of the queue manager.

The JMS Queue Manager will delete the messages on the queue after delivering it to the appropriate subscribers or after it has been on the queue the number of seconds specified in the `MaxTimeToLive` configuration parameter.

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.

Three SeeBeyond components are useful in this respect:

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

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

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

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

Retek Supplied Components

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

The following components comprise the RIB:

- Database triggers that capture application activities as they occur. These triggers are part of the specific Retek application, such as RMS. However, as part of the RIB installation and configuration, they must be enabled to capture information regarding events of interest.
- Staging tables used to hold the captured information and to maintain the publishing state of the messages.
- Publishing e*Ways that create messages from the information captured by the aforementioned Database Triggers. These publishing e*Ways are designed to publish events from a single "Message Family" and are specific to a Retek Application, such as RMS. Each RIB publishing e*Way has a collaboration that will invoke a specific stored procedure which returns the staging table information.
- Subscribing e*Ways that are used to consume messages. These are specific to a Retek Application (RMS, RCOM, RDM,…) and are designed to consume all messages from a specific message family. Each Subscribing e*Way will call a specific stored procedure used to process a specific application event message.
- Transformation Address Filters/Router (TAFR) e*Ways that transform message data and/or route messages. The TAFR acronym is a generic term. Multiple, message family specific TAFRs have been implemented. Different TAFR e*Ways may be active on different message families or on the same message family depending on the needs of an application. Not all message families require TAFRs.
- Error Hospital database tables used as a basis for storing and re-trying problematic messages.
- 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.

Additional resources

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

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

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

  Contains reference, trouble shooting and administrative information
- e*Gate Integrator Installation Guide

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

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

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

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

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

Chapter 3 – RIB component operations

This section details the message flows for a simple message and for a message undergoing a routing or filtering operation. For a more detailed description of the RIB components, see the *Retek Integration Bus Technical Architecture.* For a detailed discussion of message contents, see the *Retek 10.0 Integration Guide.*

Simple message flow

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

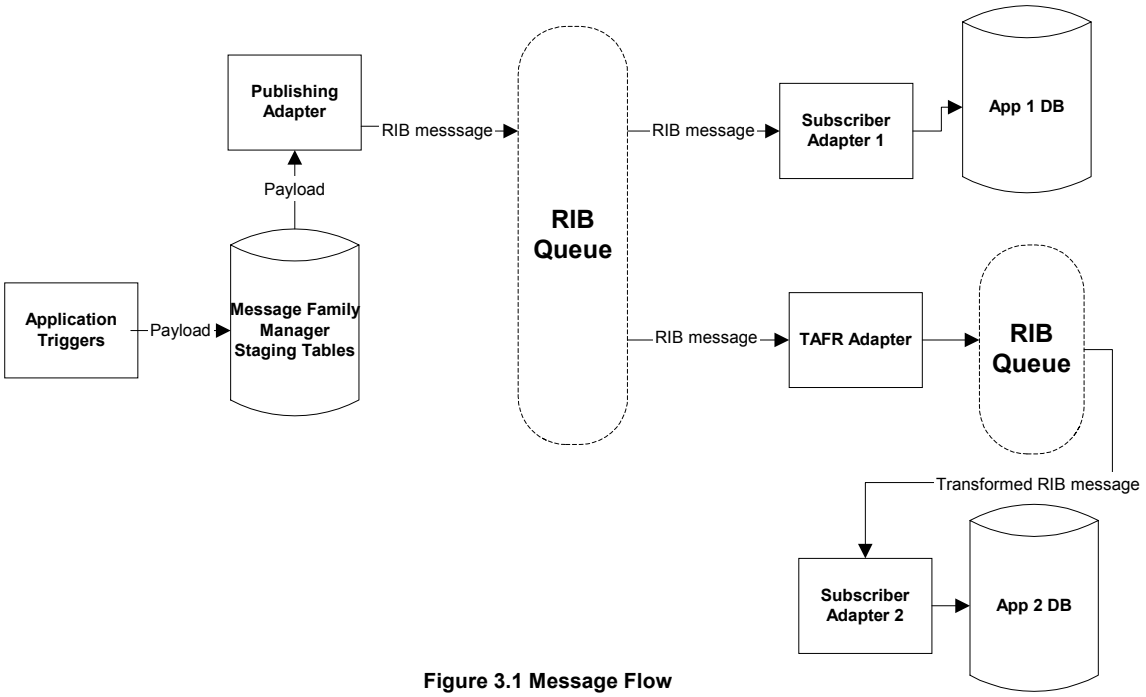


Figure 3.1 Message Flow

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

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

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

Second, a RIB Publishing e*Way is polling 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 is also polling 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 will reside on a destination specific queue/topic combination. The TAFR collaboration configuration determines the specific queue used. There must be an association of the output event type to this queue.

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

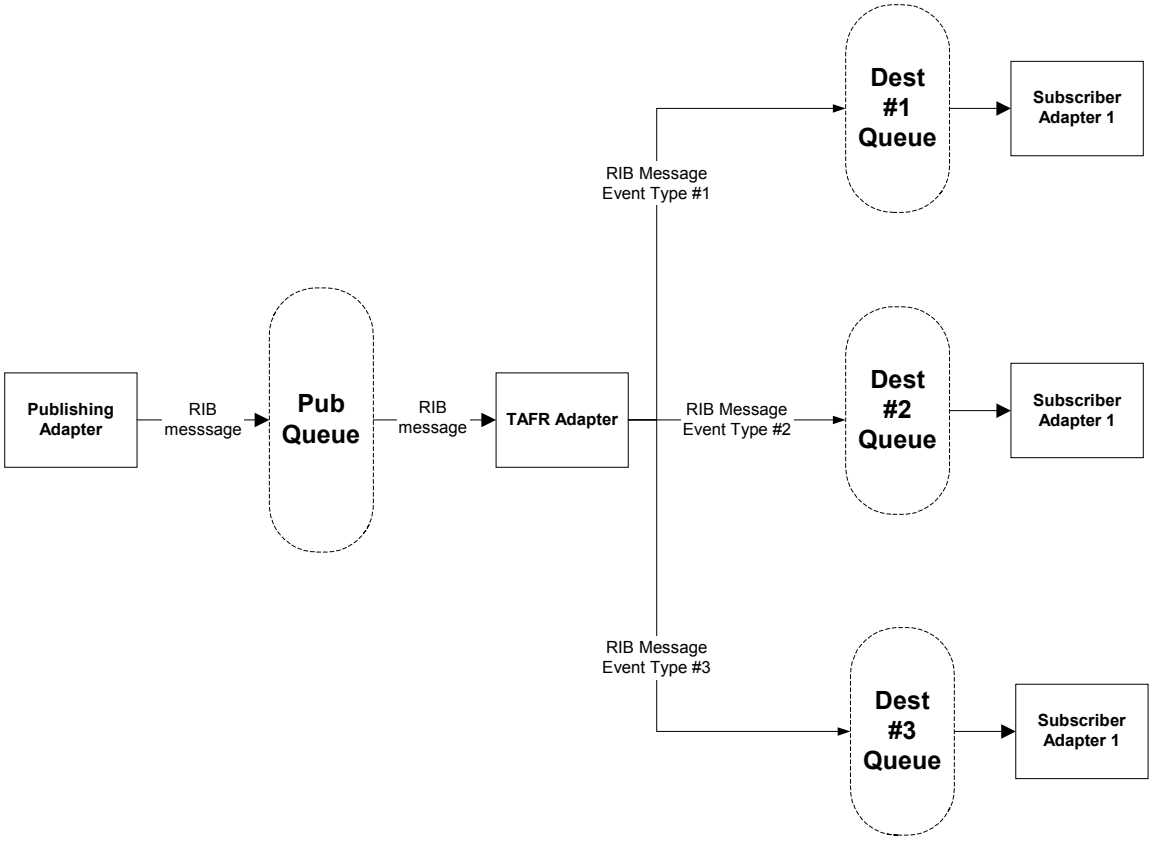


Figure 3.2 Routed Message Flow

Component failures

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

Application trigger failures

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

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

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

Publishing adapter failures

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

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

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

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

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

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

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

TAFR adapter failures

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

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

Subscribing adapter failures

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

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

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

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

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

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

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

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

In the Retek 10.0 release, a command line interface is provided to examine the contents of messages found within an Error Hospital. Error Hospital operations are detailed later in this manual.

Deployment architecture considerations

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

Retek schema integrity

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

Disk space analysis

Before the RIB is deployed to production, an analysis of the expected message traffic must be made. The Retek 10.0 Integration Guide lists all of the messages as implemented within the RIB and the conditions in which they are published. System designers use this guide to estimate expected message size and volume. From a business operations viewpoint, one should also determine the amount of time a specific subscriber is allowed to be unavailable before serious business consequences occur. This should include the maximum amount of time before a subscriber is failed-over to another system.

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

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

Intelligent Queue Managers

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

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

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

Performance motivated parallel processing

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

In the Retek 10.0 release, parallel processing considerations have been subordinated to message sequencing guarantees. In other words, the design of the system guarantees message processing is in the correct sequence as opposed to maximizing throughput.

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

Thus, additional design and implementation work is needed to support parallel processing deployments of the RIB in the 10.0 release. This work must center on creating well-defined logical channels of information, each channel responsible for a well-defined set of business entities. An example of such a logical channel would be one responsible for all of the "even numbered" purchase orders. This is similar to the Retek “Batch Thread” model. Briefly, the following changes would need to be made:

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

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

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

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

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

Chapter 4 – RIB startup and shutdown

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

Sequencing considerations

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 automatically, then only the registry and control broker's 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. On Windows systems, these components are usually installed as services.

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.

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

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

For example, for Korn or Bourne Unix shells:

```
>   . <EGATE_HOME/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. On Unix systems, this is done via the `stccb` command. On Microsoft Windows platforms, a control broker is typically installed as a service. However, 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 <EGATE_HOME>/server/egateclient.sh.

For example, for Korn or Bourne Unix shells:

```
>  . <EGATE_HOME/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 "rib100"
- 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 rib100
-un Administrator -up STC
```

- Executes *without* Access Control Lists used for authorization purposes.

11 At this point, you can display the e*Gate Monitor application to start any components not configured to be brought up automatically. This application requires a Microsoft Windows platform for execution.

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

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

RIB Message Publishing Adapters

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

The following are some of the Retek application publishing adapter names:

E*way Name	E*way Name	E*way Name
ewVendorFromRMS	ewWOInFromRMS	ewReceiptsFromRDM
ewStoresFromRMS	ewInvBalFromRMS	ewCustOrderFromRCOM
ewWHFromRMS	ewUDAsFromRMS	ewPendReturnFromRCOM
ewUDAsFromRMS	ewBannerFromRMS	ewShipMethFromRCOM
ewDiffGrpFromRMS	ewATPFromRMS	ewCOReturnFromRCOM
ewItemsFromRMS	ewRTVFromRDM	ewCOStatusFromRCOM
ewOrderFromRMS	ewInvAdjustFromRDM	ewCOStatusFromRCOM
ewOrderPhysFromRMS	ewASNInFromRDM	ewCOResFromRCOM
ewTransfersFromRMS	ewAppointFromRDM	ewCOSaleFromRCOM
ewAllocFromRMS	ewASNOutFromRDM	
ewDiffsFromRMS	ewCustReturnFromRDM	

RIB Message Subscribing Adapters

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

The following are some of the Retek application subscribing adapter names:

E*way Name	E*way Name	E*way Name
ewVendorToRCOM	ewAppointToRMS	ewASNInToRMS
ewVendorToRDM	ewReceiptsToRMS	ewASNInToRDM
ewStoresToRCOM	ewSOStatusToRMS	ewRTVToRCOM
ewWHToRCOM	ewSOStatusToRCOM	ewRTVToRMS
ewLocationsToRDM	ewASNOutToRMS	ewInvBalToRMS
ewCOReturnToRMS	ewASNOutToRCOM	ewUDAsToRDM
ewItemsToRCOM	ewInvAdjustToRMS	ewCustReturnToRCOM
ewItemsToRDM	ewDiffGrpToRCOM	ewPendReturnToRDM
ewOrderPhysToRDM	ewDiffsToRCOM	ewCOStatusToRMS
ewStockOrderToRDM	ewDiffGrpToRDM	ewCOResToRMS
ewAppointToRMS	ewDiffsToRDM	ewCOSaleToRMS
ewATPToRCOM		

TAFR adapters

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

The following are some of the Retek application subscribing adapter names:

Eway Name	Eway Name	Eway Name
ewStoresToLocationsFromRIB	ewTransfersToStockOrderFromRIB	ewASNOutToASNOutCOFromRIB
ewWHToLocationsFromRIB	ewTransfersToStockOrderWHFromRIB	ewASNInToASNInWHFromRIB
ewStoresToLocationsFromRIB	ewTransfersToStockOrderFromRIB	ewWOInToWOInWHFromRIB
ewWHToLocationsFromRIB	ewAllocToStockOrderFromRIB	ewUDAsToUDAsLVFromRIB
ewShipMethToLocationsFromRIB	ewCustOrderToStockOrderFromRIB	ewASNOutToASNOutCOFromRIB
ewItemsToItemsSPFromRIB	ewSOStatusToSOStatusCOFromRIB	

RIB Error Hospital start/stop

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

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

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

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

Chapter 5 – Preventative maintenance tasks

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

Log files

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

Error, trace, 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 <EGATE_HOME>/client/logs, where <EGATE_HOME> 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.

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

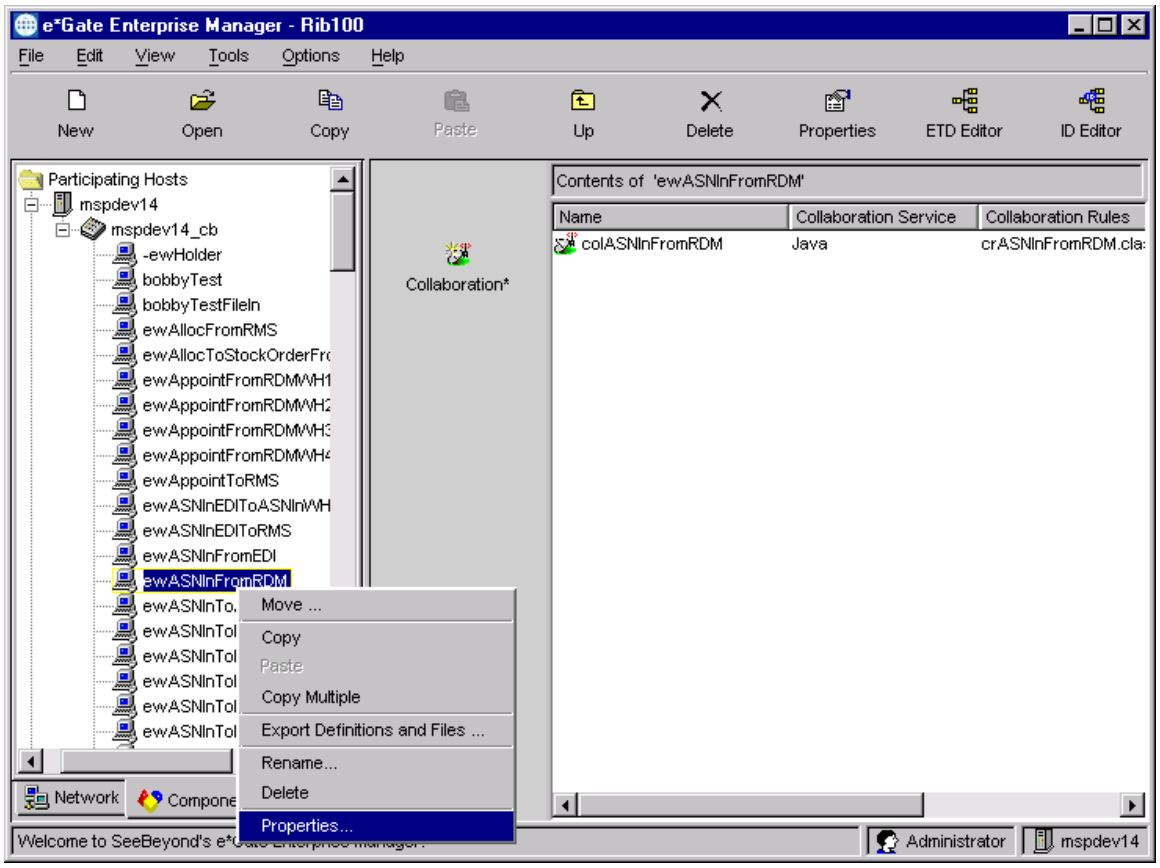


Figure 5-1: Selecting an e*Way from the e*Gate Enterprise Manager

2 Right click on the e*Way.

3 Select Properties. The Properties window is displayed:

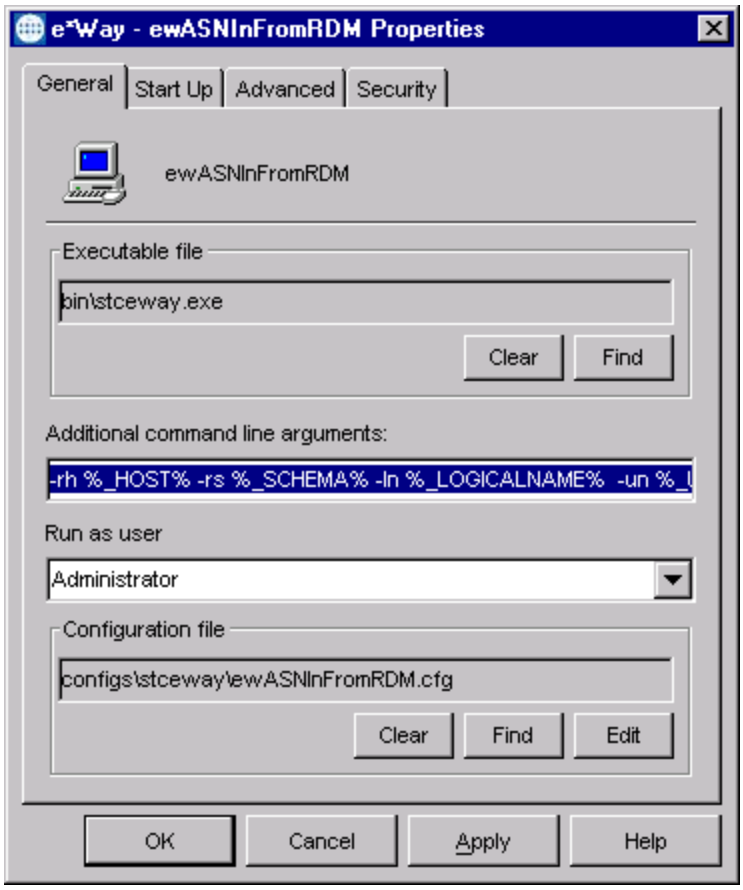


Figure 5-2: e*Way Properties window (General tab)

4 Click on the Advanced tab.

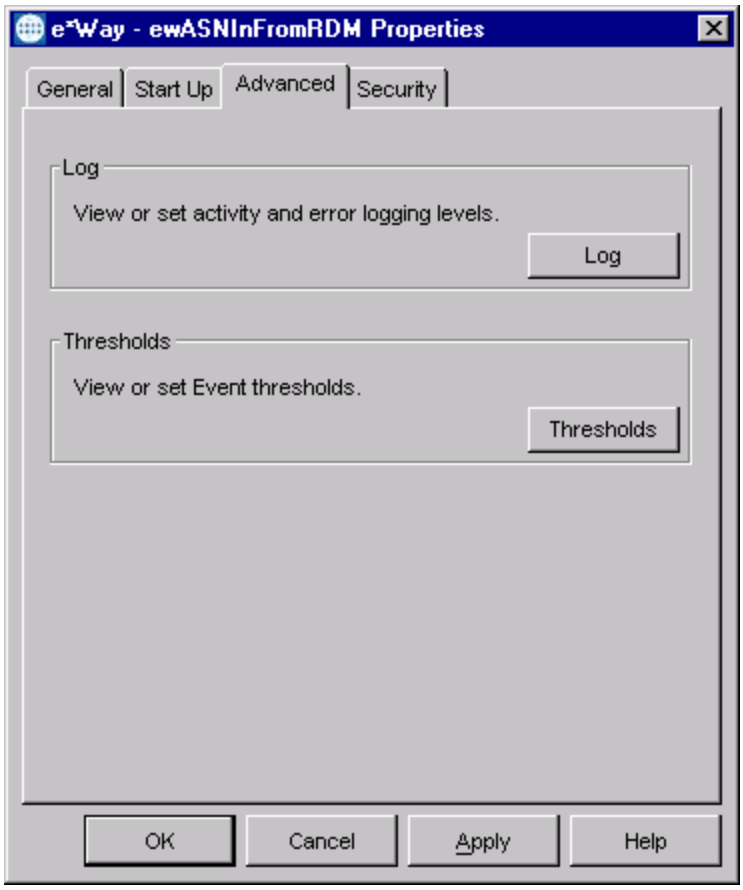


*Figure 5-3: e*Way Properties window (Advanced tab)*

5 Click **Log** .

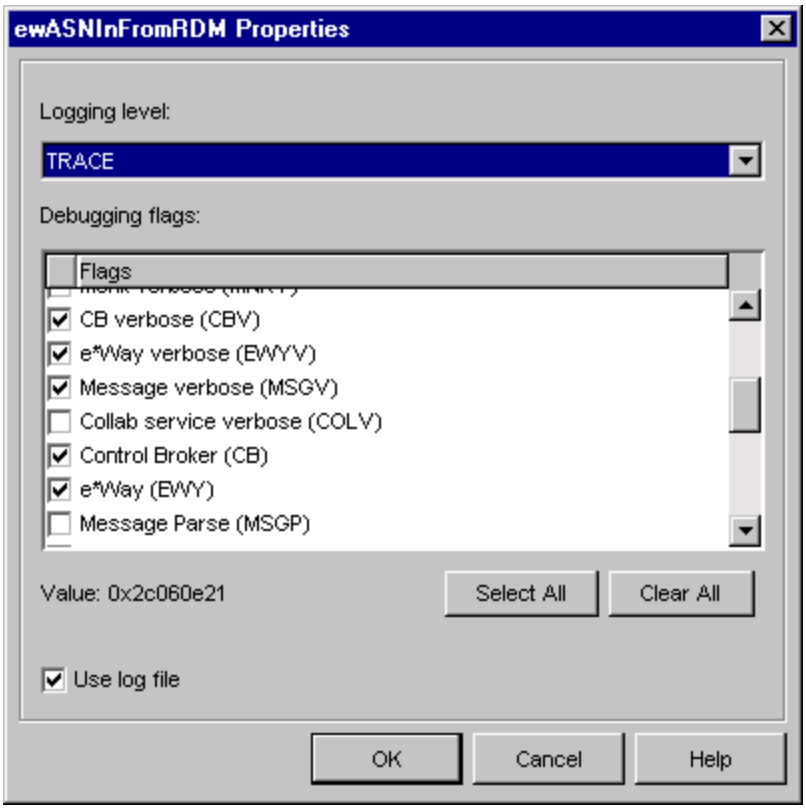


Figure 5-3: e*Way Logging window

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

Over 25 different areas are available for logging.

To log RIB Adapter-created messages:

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

7 In the Logging File field, select TRACE.

8 Select the Use Log file check box.

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

XA Transaction Log Files

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

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

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

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

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

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

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

MFM staging tables

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

The code that wrappers access to these staging tables is known generally as the Message Family Managers (MFMs). The MFM implements the interfaces for extracting the data as procedures found within an Oracle database package. For more information on MFMs in general, see the Retek Integration Bus Technical Architecture Guide. For information about a specific MFM, see the Retek 10.0 Integration Guide.

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

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

Error Hospital

The Error Hospital uses three database tables for stable storage. Because a message can only be retried a set number of times, messages may permanently reside in the Error Hospital and require manual intervention before they can be deleted.

Error Hospital tables should be monitored on a daily basis. See Chapter 6 – Message Error Handling for more details.

SeeBeyond Tools

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

e*Gate Monitor and Queue Administration Tools

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

The e*Gate monitor is a GUI that can display all components found in a specific schema. Additional GUI applications are accessible from the e*Gate monitor. There is a queue monitor for SeeBeyond standard “Intelligent Queues” (the IQ Administrator) and one for JMS queues (the JMS Administrator).

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

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

e*Gate Enterprise Manager

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

Command Line Utilities

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

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

stcinstd

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

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

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

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

stcregutil

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

stcaclutil

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

stciqutil

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

stcutil

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

stccmd

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

```
? - list available commands

activate <component name> - activate element operations

attachiq <IQ name> - IQ to bring up

cls [cmd|stat] - clear window

debug <component name> [flag] - show or change an
element's debug flags

detachiq <IQ name> - IQ to detach

exit - exit stccmd.exe

getres [-b<begin date (mm/dd/ccyy)> | -e<end date
(mm/dd/ccyy)] - show resolved notifications

getstatus [-b<begin date (mm/dd/ccyy)> | -e<end date
(mm/dd/ccyy)] - show status-type notifications

getunres [-all | -a] - show unresolved notifications

help <command> - on-line help

history - list command history

list     [
all | monitors {-m} | alertors {-a} | iq {-i} | control
{-c}
| notif {-n} [flush | all
| -b<begin  date (mm/dd/ccyy)>  [-e<end date
(mm/dd/ccyy)>]
| +r | -r | -i<notification number> | <component name>
]

 quit - exit stccmd.exe

 reload <component name> [hard] - reload configuration

 resolve <notification number> - indicate that a
notification has been resolved

 sequence <component name> [value] - show or change
sequence number

 shell <shell command> - run an external command

 shutdown <component name> - controlled module shutdown

 shutdownall <shutdownall> - controlled modules shutdown

 start <component name> - start or restart module

 startall <startall> - start or restart all modules

 status <component name> - show status

 suspend <component name> - suspend operations
```

```
version <component name> - Show version
```

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

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

```
stccmd.exe -rh egate_main -rs Rib100 -cb egate_cb -un
Administrator -up STC -cmd list all
```

Where `egate_main` is the registry host, Rib100 is the schema name, `egate_cb` is the control broker to connect to, `Administrator` is the e*Gate user name to use, and `STC` is the password for the Administrator user.

Chapter 6 – Message error handling

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

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

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

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

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

If a message contains invalid data and there are three subscribers for this message family, then each subscriber will store a copy of the message in an Error Hospital and re-publish the message to the queue. In order to accomplish this, the event type of the re-submitted message is changed. The new event type is specific to the subscribing application/message family combination. For example, if the original message was published with the event type of etASNInFromRMSWH1, then the event type for re-tries might be etASNInFromRMSWH1Retry.

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

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

Error Hospital components

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

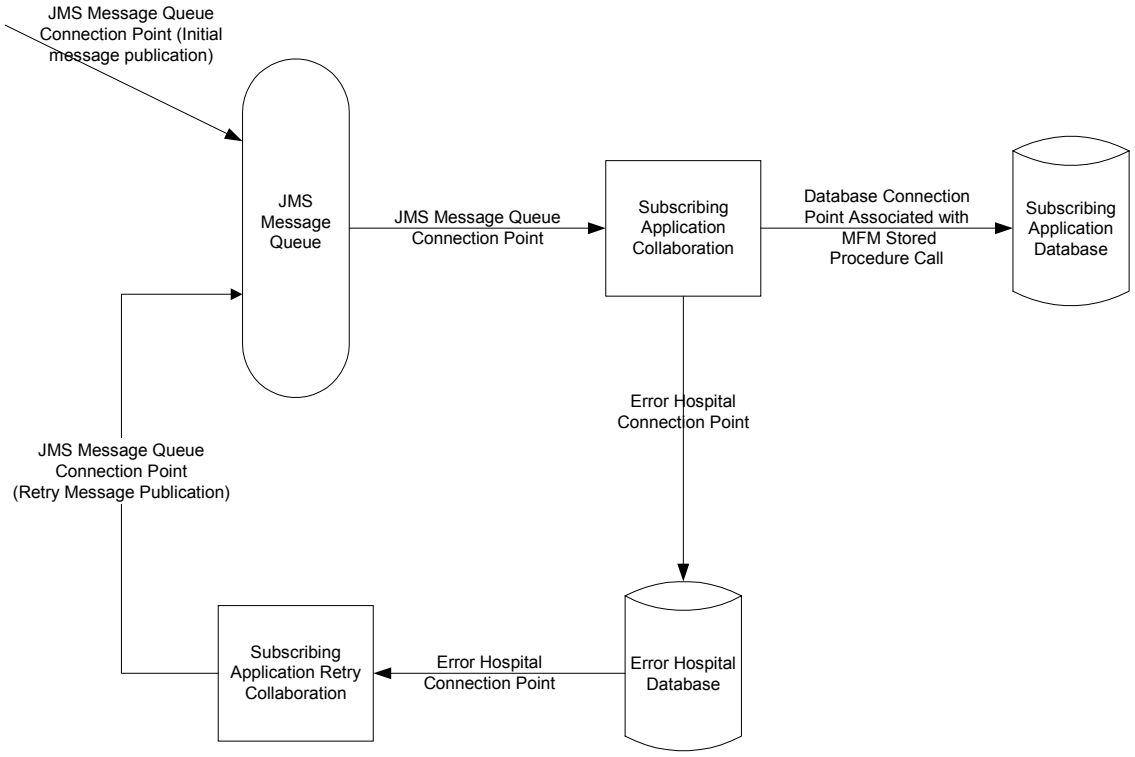


Figure 6-1 Connection Points used at a subscriber.

The following tables are used to store message information within the Error Hospital:

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

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

Error Hospital configuration parameters and properties

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

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

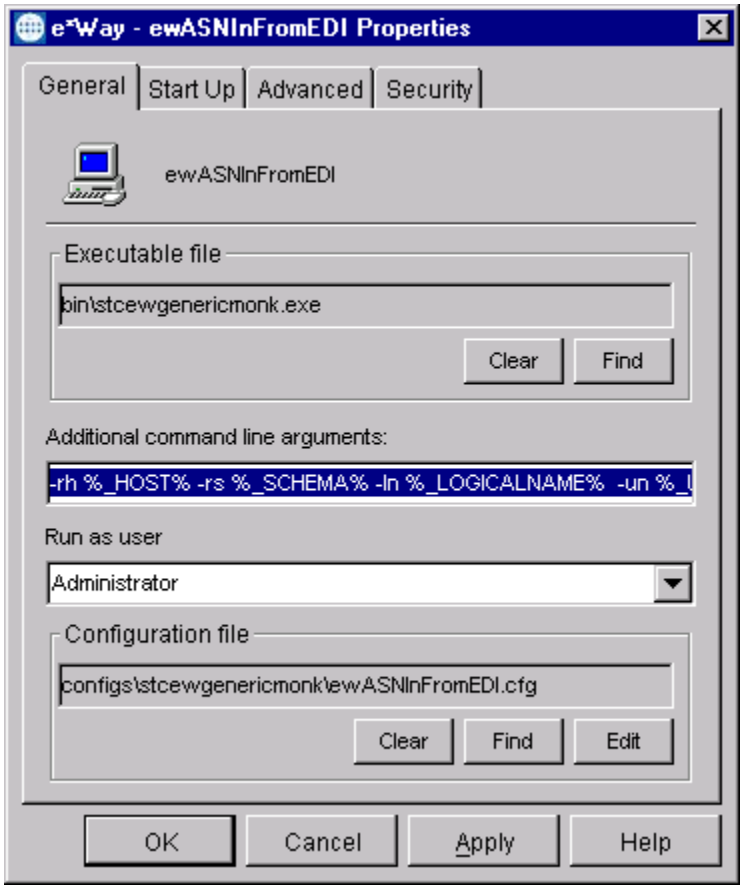
To access the adapter configuration:

1 Open the SeeBeyond Enterprise Manager.

2 Select an option:

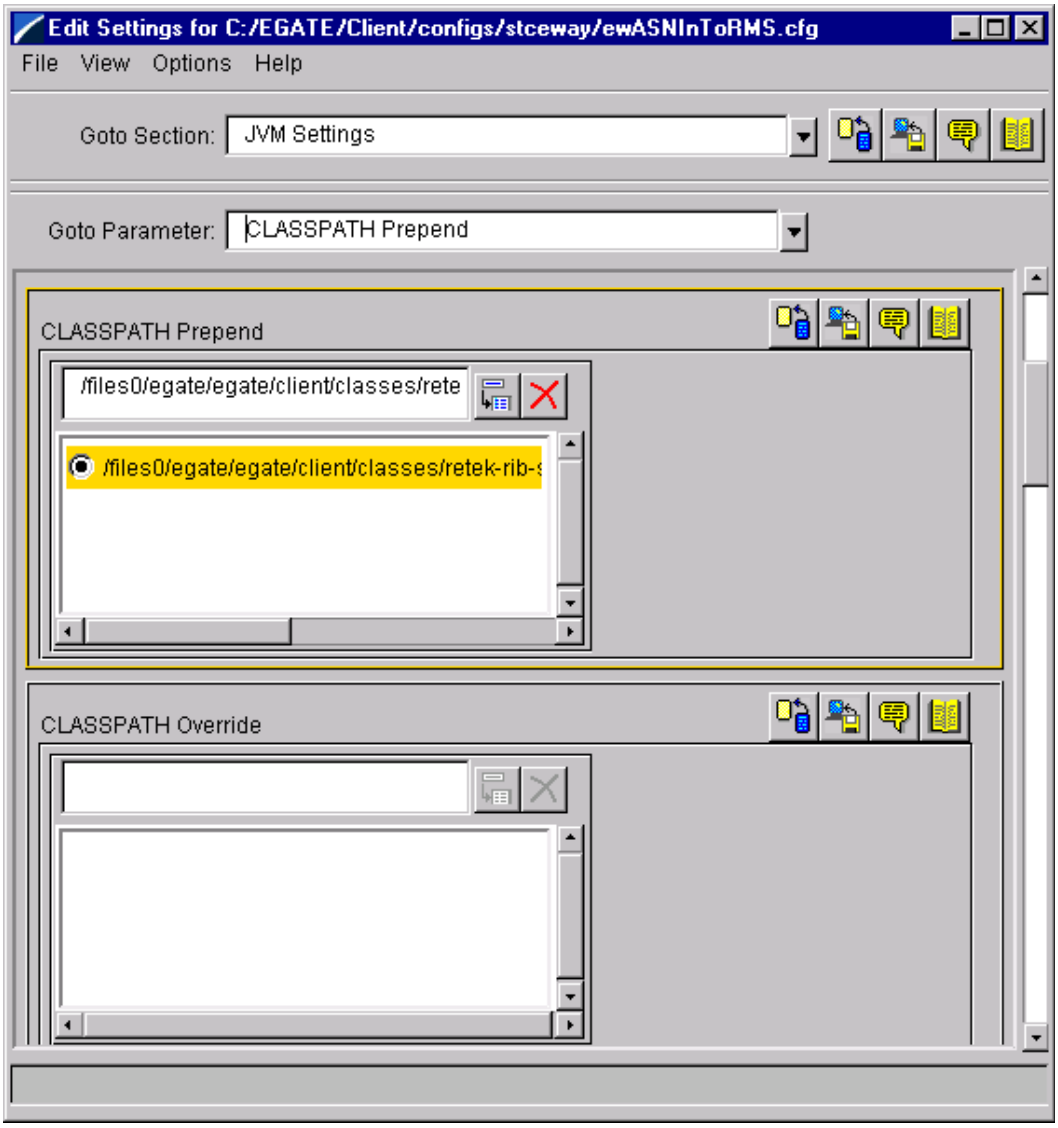
- Right click on the appropriate subscribing e*Way and select Properties.
- Select the appropriate subscribing e*Way and then click the Properties toolbar button.

The e*Way Properties dialog box is displayed.



*e*Way Properties dialog box*

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



Configuration file edit window

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

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

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

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

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

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

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

Error Hospital activities

This section details activities one may perform on the Error Hospital from the Error Hospital command line utility. This Java application lets you:

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

The Error Hospital command line utility is a single Java class that is executed or wrappered by a set of shell scripts (Unix platform) or BAT files for the Windows NT or Windows 2000 environment. This Java class requires the presence of a properties file, `hospital-admin.properties`, in the user's home directory.

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

Hospital admin command line utility set up

There hospital-admin.properties file and the hospital-admin.env file must be manually set up before the command line utility can be used.

Setting up hospital-admin.properties

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

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

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

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

```
hospital.gui.prop.dbUser=retek_user

hospital.gui.prop.dbPwd=retek_password

hospital.gui.prop.dbUrl=jdbc:oracle:thin:@HSP_DB_HOST:15
21:hsp_SID

hospital.gui.prop.dbDriverClass=oracle.jdbc.driver.Oracl
eDriver
```

Setting up hospital-admin.env

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

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

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

  ```
  CLASSPATH=/files0/egate/egate/client/classes/retek-rib-support.jar:/files0/egate/egate/client/ThirdParty/oracle/classes/classes12.zip:/files0/egate/egate/client/etd/etdRibMessageEnvelope.jar:/files0/egate/egate/client/classes/stcjcs.jar
  ```

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

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

Error Hospital admin command line scripts

All Error Hospital administration is done via the Java class:

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

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

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

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

Command	Parameters	Description
		• -f <family> lists only those message numbers belonging to the specified message families • -g <type> lists only those message numbers that belong to messages of the specified type • -i <id> lists only those message numbers that apply to the specified ID. These identify a specific business object, such as a Purchase Order or ASN. • -q <inQueue> lists only those message numbers that are believed to be enqueued in the integration bus at the current time. A value of 0 or "false" implies the message only exists in the Error Hospital, a value of 1 or "true" implies that the message is thought to have been published for another attempt to process it. • -r <retry> lists only those messages according to their retry status. The <retry> specification of 0 or "false" lists those not eligible for retry and marked ready for delete; a value of 1 or "true" lists those eligible for retry and not ready to be deleted. All parameters are optional. Multiple parameters produce the intersection of their independent results. (For example, –f Family and –l Location lists all messages in family "Family" belonging to location "Location".)
deletemsg	-m <messageID> -p <propertiesFile>	Marks the message ready for deletion. The message will be deleted when the retry collaboration next awakens. The –m switch is mandatory and must contain the message number of the message to delete.
readmsg	-m <messageID> -F <outputFileName> -p <propertiesFile>	Retrieves the payload contents for message <messageID> and writes it out to the file <outputFileName>. The –m switch is mandatory and must contain the message number of the

Command	Parameters	Description
		message to read.
updatemsg	-m <messageID> -f <inputFileName> -p <propertiesFile>	Replaces the message payload for the given message with the contents of the file. No validation of the file contents is performed until the subscribing adapter processes the data. The –m switch is mandatory and must contain the message number of the message to update.
stopmsg	-m <messageID> -p <propertiesFile>	Stops further attempts to retry the message. The –m switch is mandatory and must contain the message number of the message to stop retrying.
retrymsg	-m <messageID> -p <propertiesFile>	Flags the message so one additional attempt is made to process the message. The –m switch is mandatory and must contain the message number of the message to retry.

Hospital Administration command line examples

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

Listing all messages in an Error Hospital:

```
> querymsg

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

java HospitalAdminCmdLine -a query

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

Number of messages selected: 159

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

SUCCESS

>
```

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

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

```
> querymsg -l ewASNOutToRCOM%

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

java HospitalAdminCmdLine -a query -l ewASNOutToRCOM%

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

Number of messages selected: 15

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

SUCCESS

>
```

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

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

```
> querymsg -f asnout

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

java HospitalAdminCmdLine -a query -f asnout

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

Number of messages selected: 23

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

SUCCESS
```

Reading the message payload XML into a file

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

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

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

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

read Message: 2947
```

```
SUCCESS

> cat /tmp/message_2947.xml

<?xml version="1.0" encoding="UTF-8"?><!DOCTYPE
POReceiptDesc SYSTEM
"http://mspdev09:8109/dtdtst/POReceiptDesc.dtd"><PORecei
ptDesc><dc_dest_id>1</dc_dest_id><appt_nbr>500000301</ap
pt_nbr><po_nbr>10610</po_nbr><document_type>P</document_
type><item_id>100614114</item_id><unit_qty>8</unit_qty><
receipt_xactn_type>R</receipt_xactn_type><receipt_date><
year>2002</year><month>03</month><day>08</day><hour>16</
hour><minute>47</minute><second>11</second></receipt_dat
e><receipt_nbr>500000291</receipt_nbr><asn_nbr>ASN-IT-
RECEIPT-
19</asn_nbr><dest_id>1000000014</dest_id><container_id>A
SN-IT-REC-19-
CID001</container_id><distro_nbr>1000001911</distro_nbr>
<distro_doc_type>A</distro_doc_type><to_disposition>WIP<
/to_disposition><from_disposition></from_disposition><to
_wip>MXDSKU</to_wip><from_wip></from_wip><to_trouble></t
o_trouble><from_trouble></from_trouble><user_id>ZZRUDEJ<
/user_id></POReceiptDesc>

>
```

Updating the message payload from a file:

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

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

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

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

update Message: 2947

SUCCESS

>
```

Marking a message ready for deletion:

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

```
> deletemsg -m 2155

java HospitalAdminCmdLine -a delete -m 2155

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

delete Message: 2155

SUCCESS

>
```

Manually querying message information from the Error Hospital

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

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

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

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

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

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

To display the failure history of a specific message

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

To display the message numbers for a particular message type

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

Columns in the RIB_MESSAGE table

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

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

Error Hospital log entries

The Error Hospital software contains trace statements for monitoring its execution. These statements will be logged to the SeeBeyond e*Way log files. More verbose logging of hospital operations is available if the e*Way's Log settings have been set to log 'e*Way (EWY)' messages and the log level has been set to "DEBUG" or "TRACE". These settings can be modified by using the e*Gate Enterprise Manager application.

The standard log file will be used (<EGATE_HOME>/client/logs/<EWAY_NAME>.log).

Unfortunately, once this logging level has been set, the adapter will log many more messages from other e*Way components. The following Unix command may be used to see just the Error Hospital messages (from either the subscribing collaboration or the retry collaboration):

```
egrep  'Hospital(Controller|Persistence) '
<LOG_FILE_NAME>
```

where `<LOG_FILE_NAME>` is the name of the log file.

Note: There *is* a space both before and after the second single-quotation mark.

Creating additional error hospitals

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

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

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

Chapter 7 – RIB component configuration

This section details configuration issues and options with the RIB.

Oracle database triggers

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

RIB property file

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

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

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

Multichannel_ind property

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

```
multichannel_ind = MPHYS
```

Legal values for this property are:

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

SeeBeyond e*Way configuration files

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

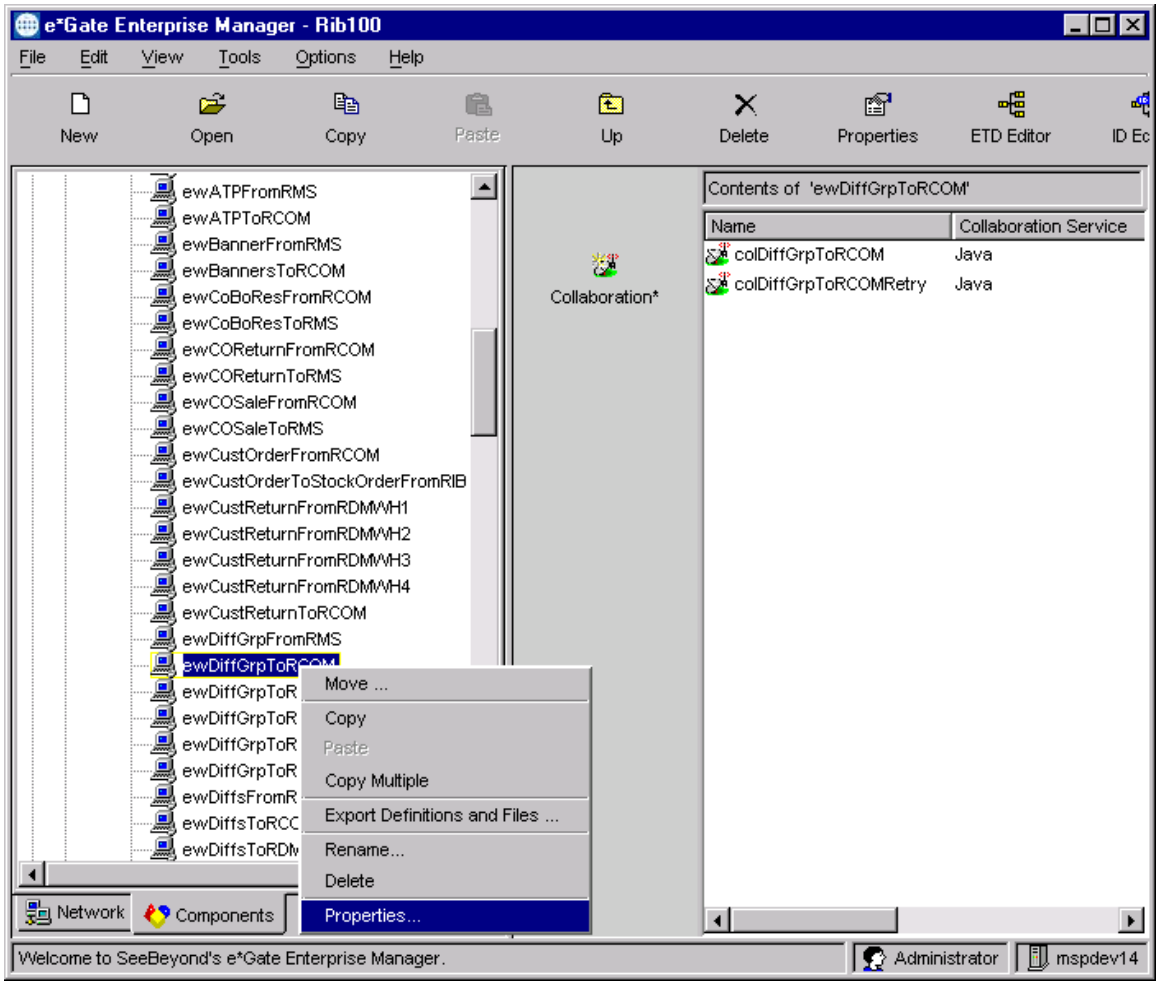


Figure 7-1: Right-click on e*Way in e*Gate Enterprise Manager

e*Way property and configuration files

Figure 7-1 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.

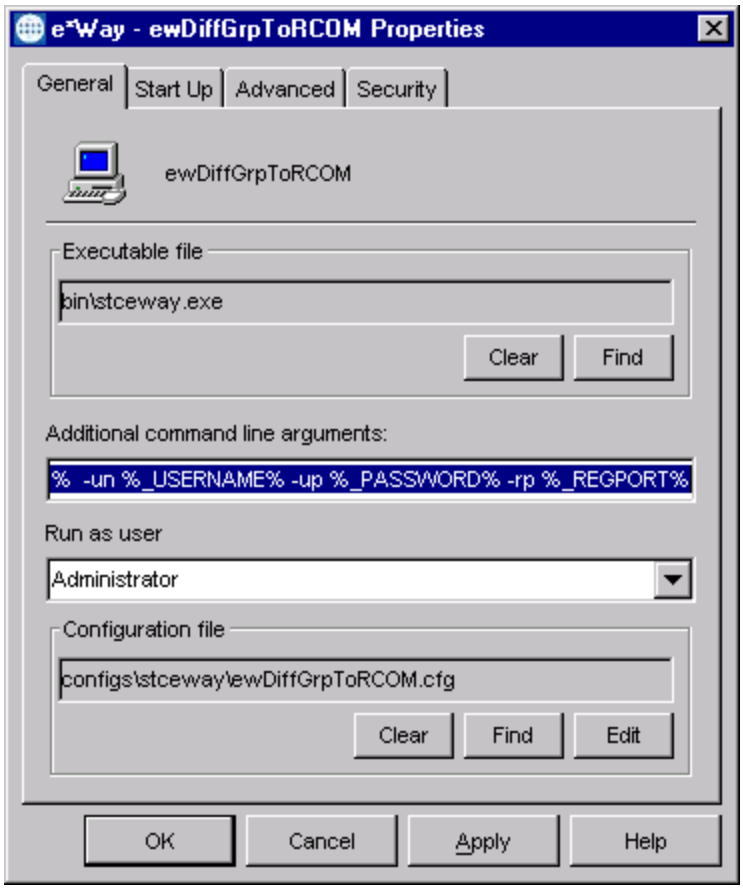


Figure 7-2: e*Way Properties Window

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

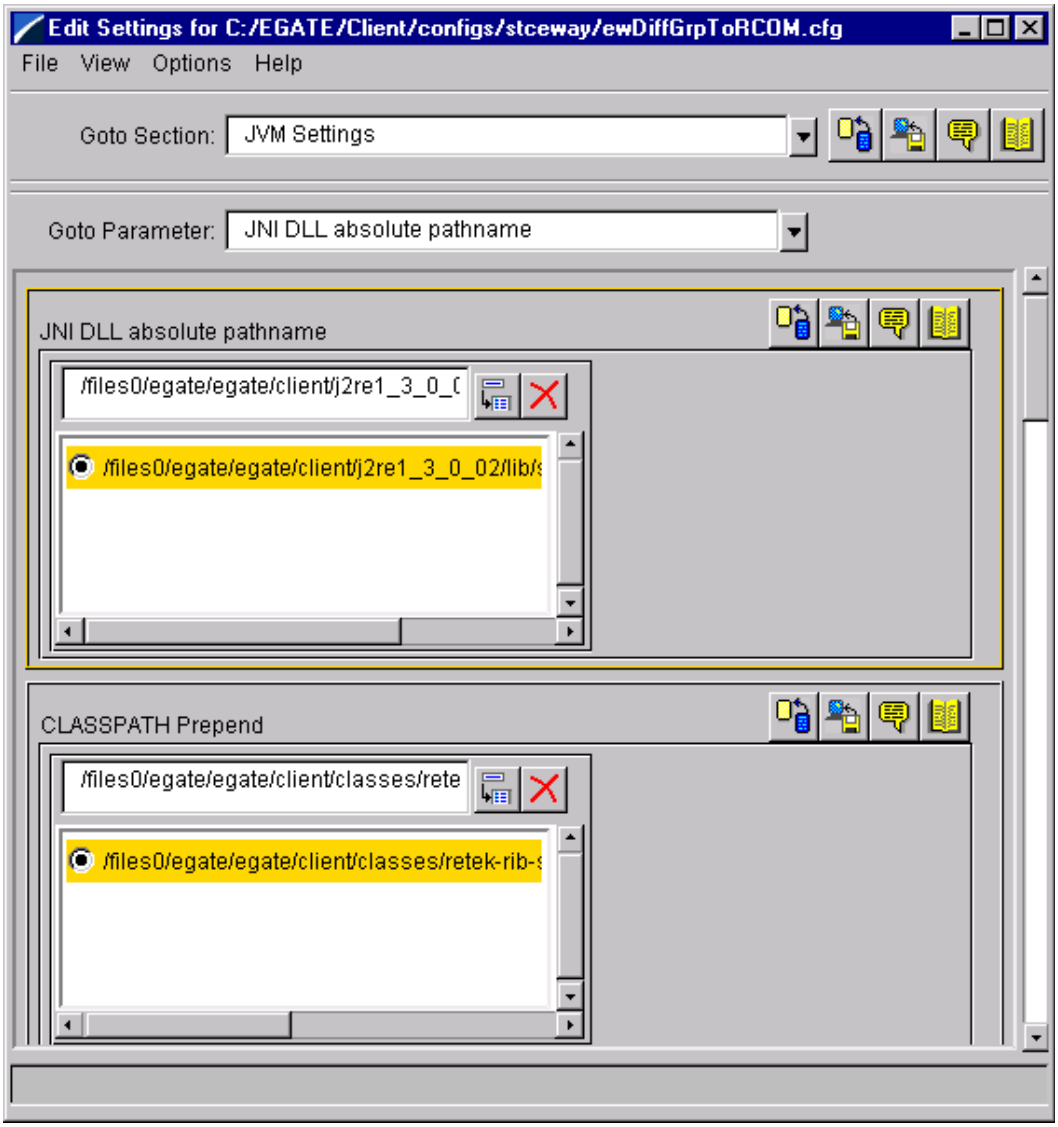


Figure 7-3: e*Way Configuration Edit Window

The configuration for this e*Way is the file <EGATE_HOME>\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:

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

<EGATE_HOME>/client/j2re1_3_0_02/lib/sparc/client/lib
jvm.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. The RIB Support JAR file is typically found at

  ```
  <EGATE_HOME>/client/classes/retek-rib-support.jar
  ```

  while the RIB Properties file is typically found at

  ```
  <EGATE_HOME>/client/classes/rib.properties
  ```

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

  ```
  /opt/egate/client/classes/retek-rib-support.jar:
         /opt/client/classes/rib.properties
  ```

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

  ```
  C:\egate\client\classes\retek-rib\support.jar;
  c:\egate\client\classes\rib.properties
  ```

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.

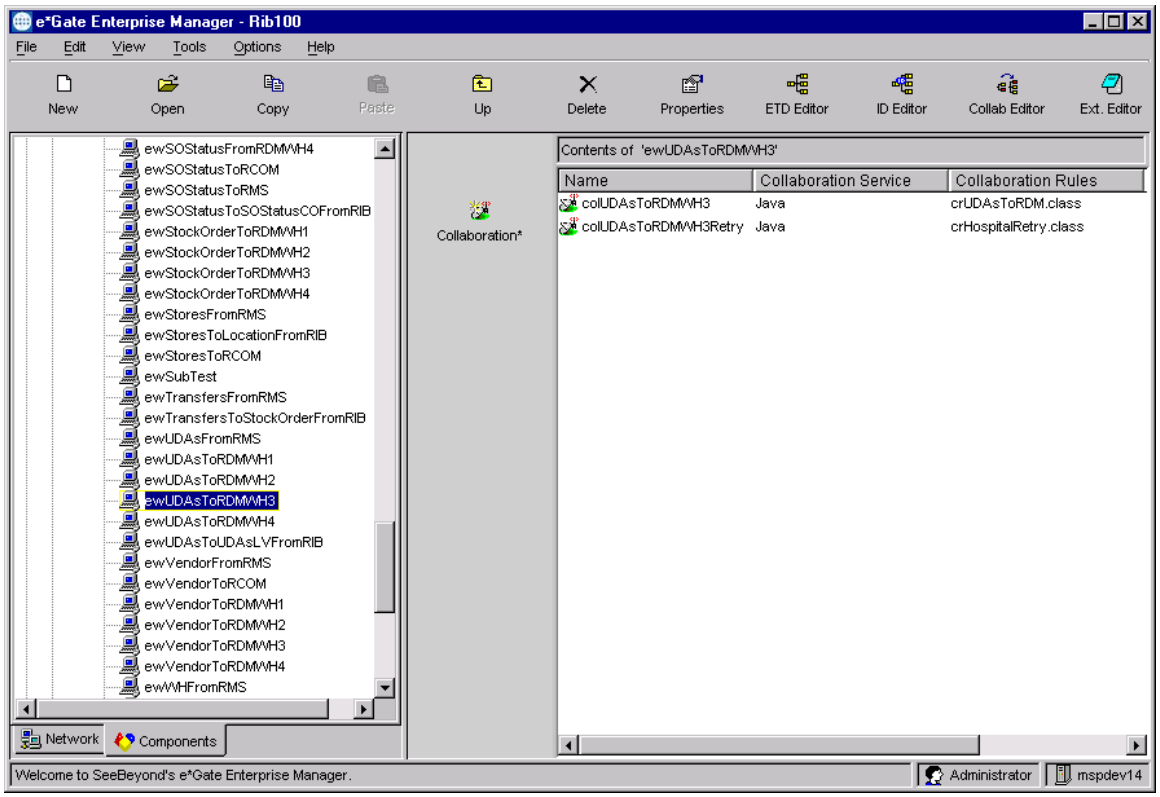


Figure 7-4: Main e*Gate window when RIB e*Way selected

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:

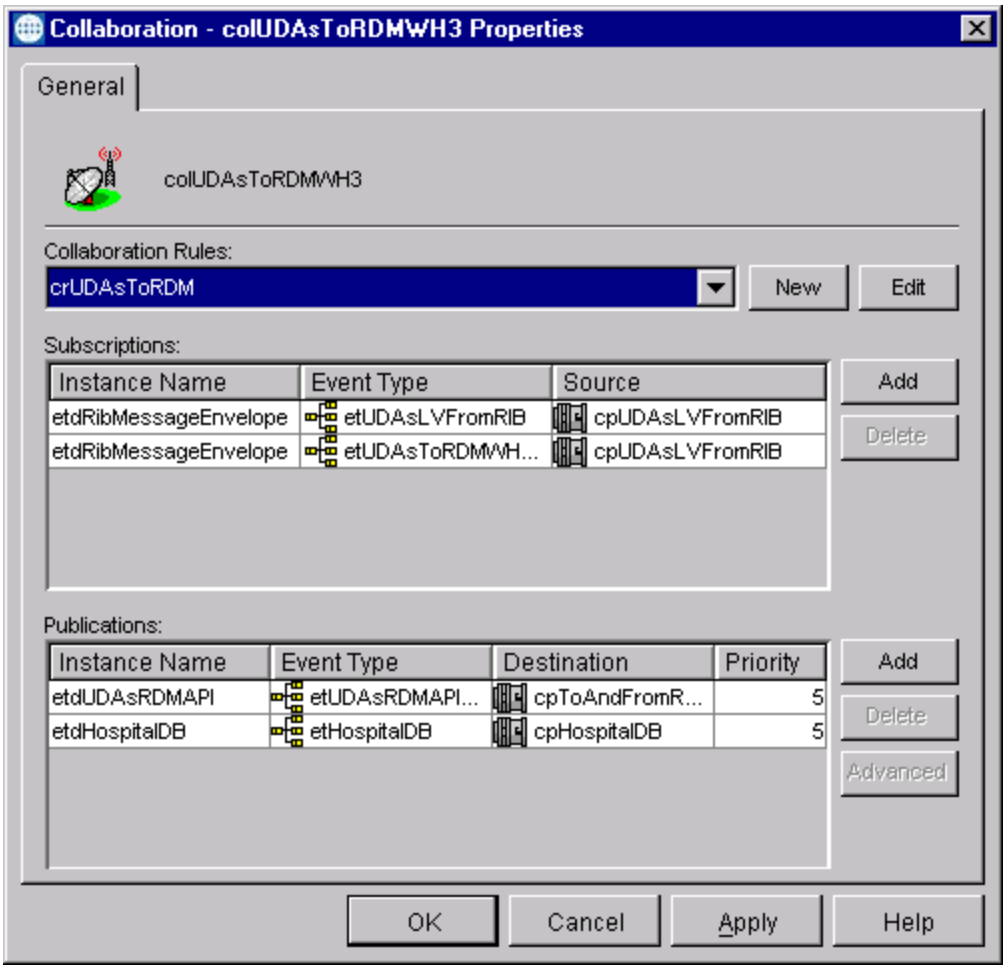


Figure 7-5: "subscribing" e*Way collaboration properties

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:

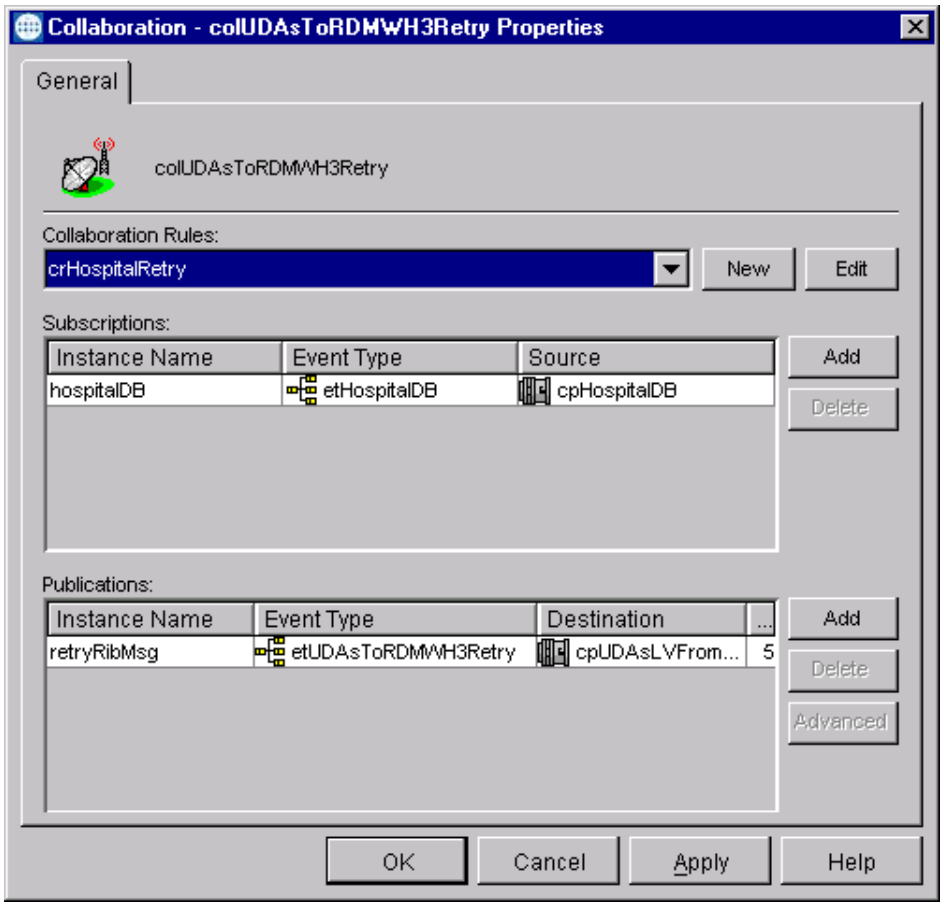


Figure 7-6: "Retry" collaboration properties

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.

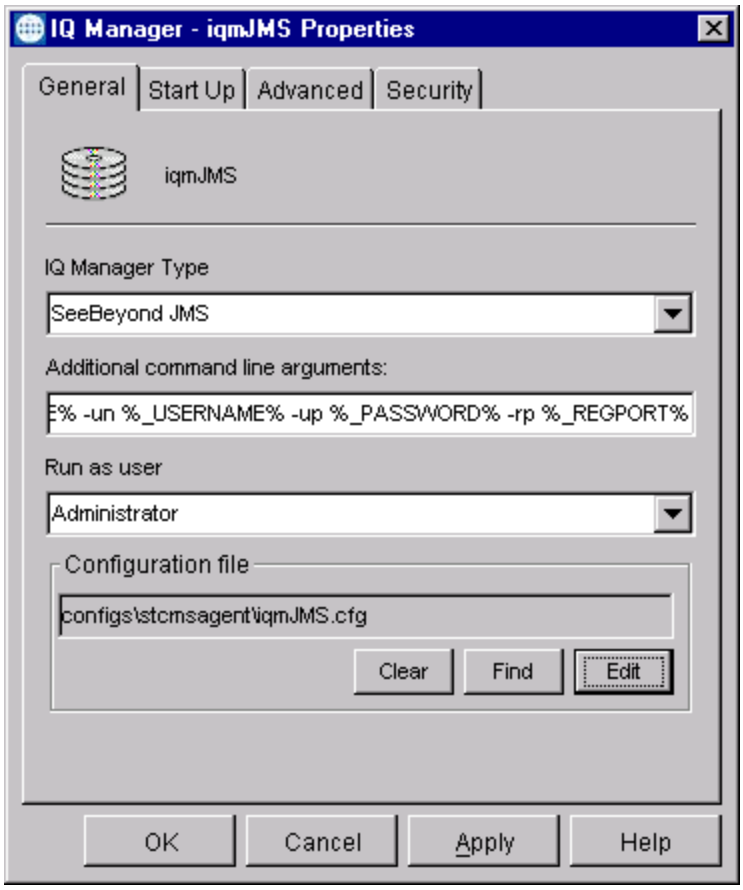


Figure 7-7: JMS IQ Manager Properties Window

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.

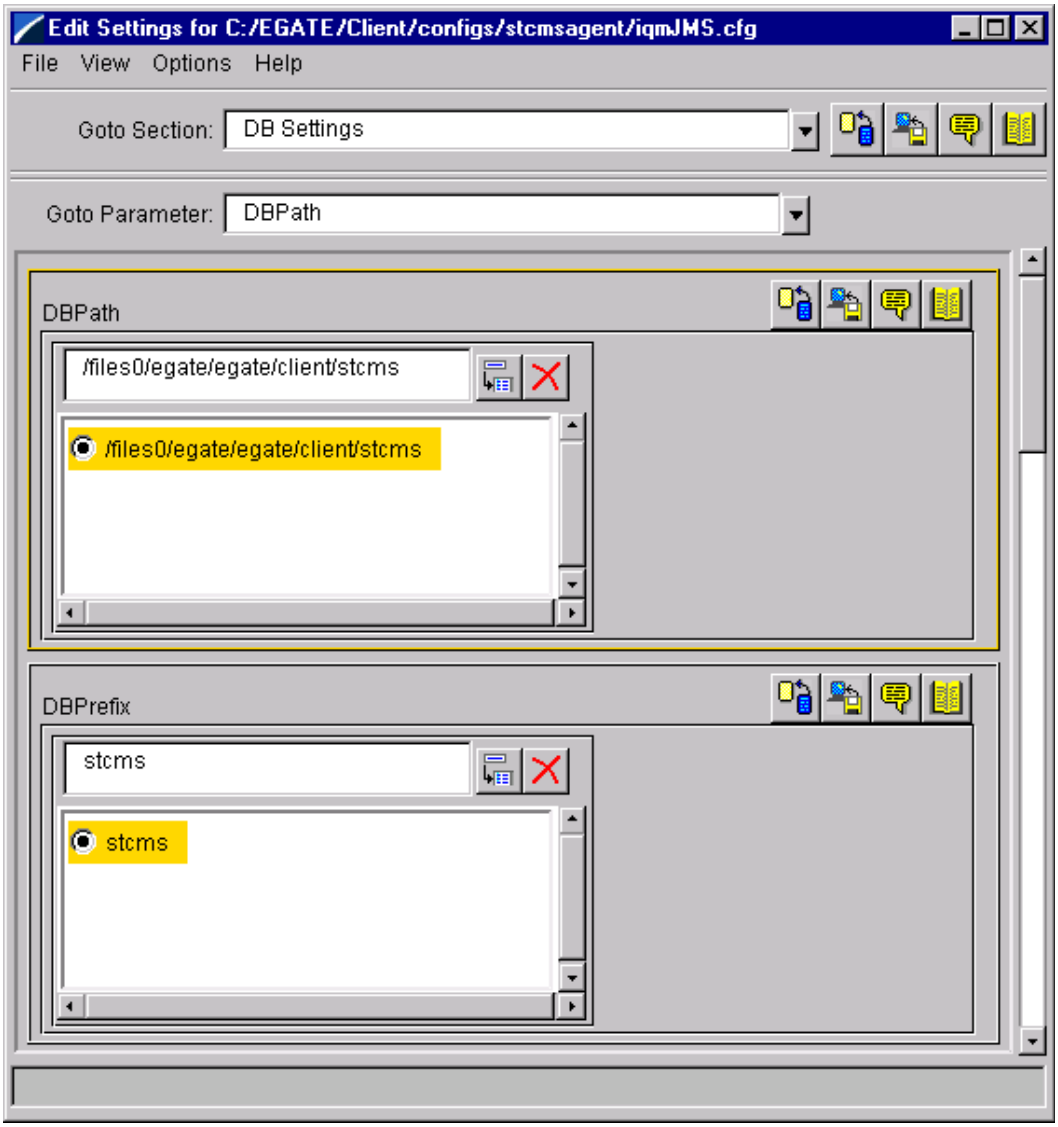


Figure 7-8: JMS IQ Manager Configuration Edit window

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.

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:

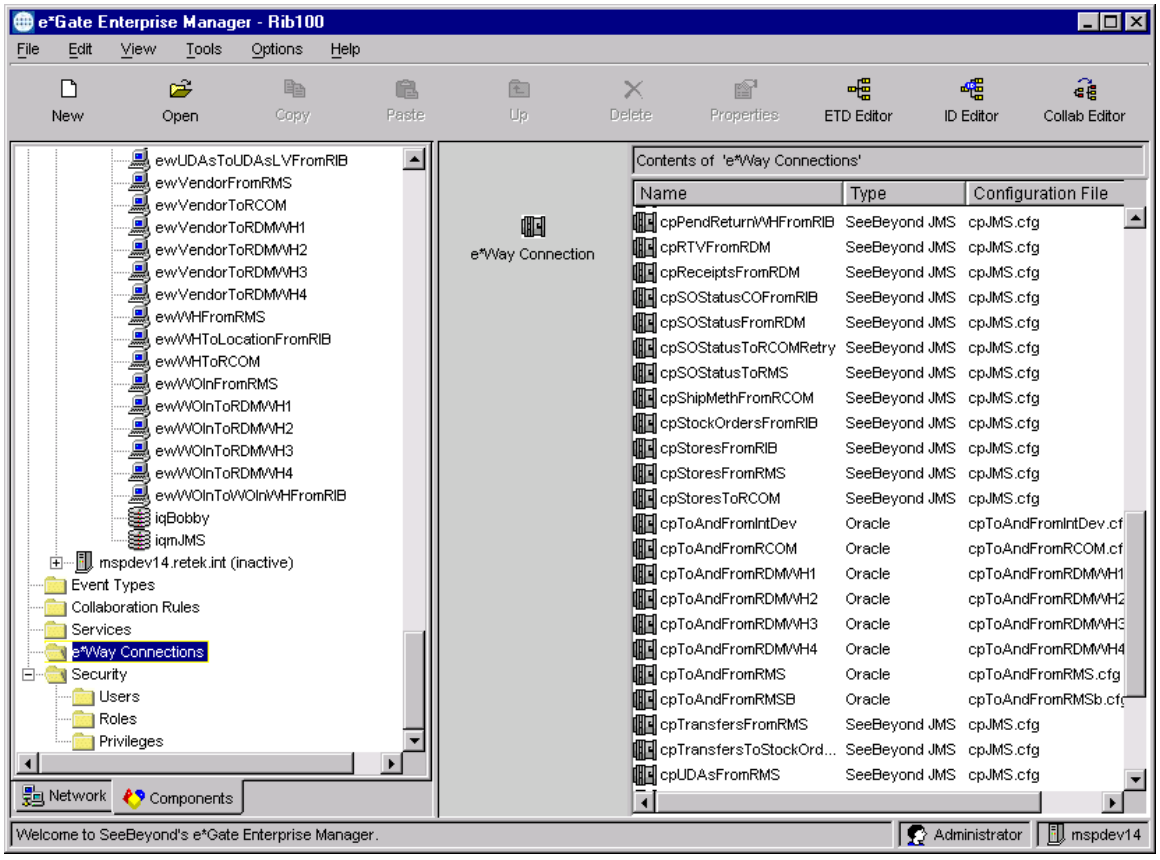


Figure 7-9: e*Gate Enterprise Manager with e*Way Connections folder selected

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).

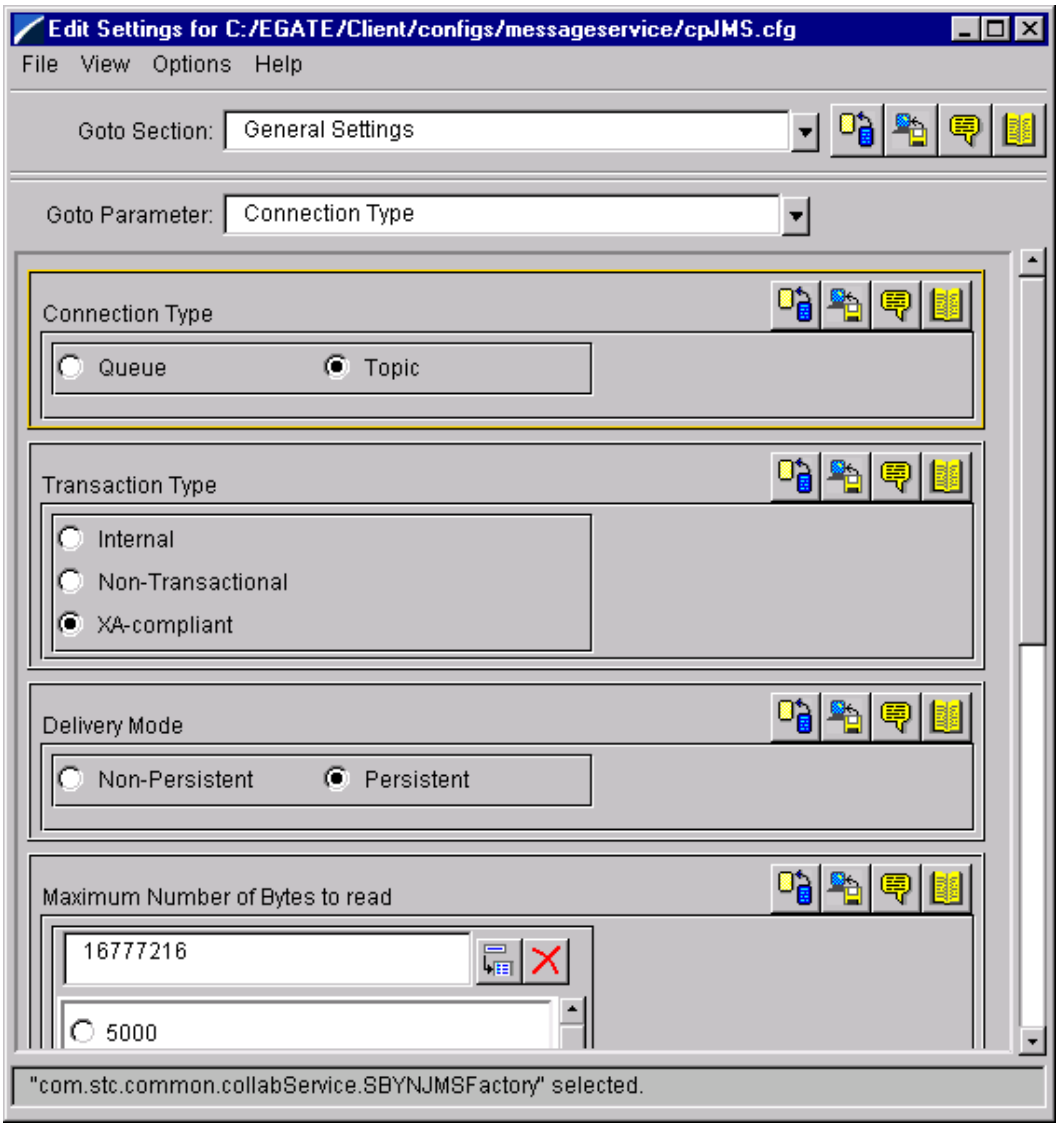


Figure 7-10: JMS Connection Point Configuration Edit window

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".
- F**actory 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:

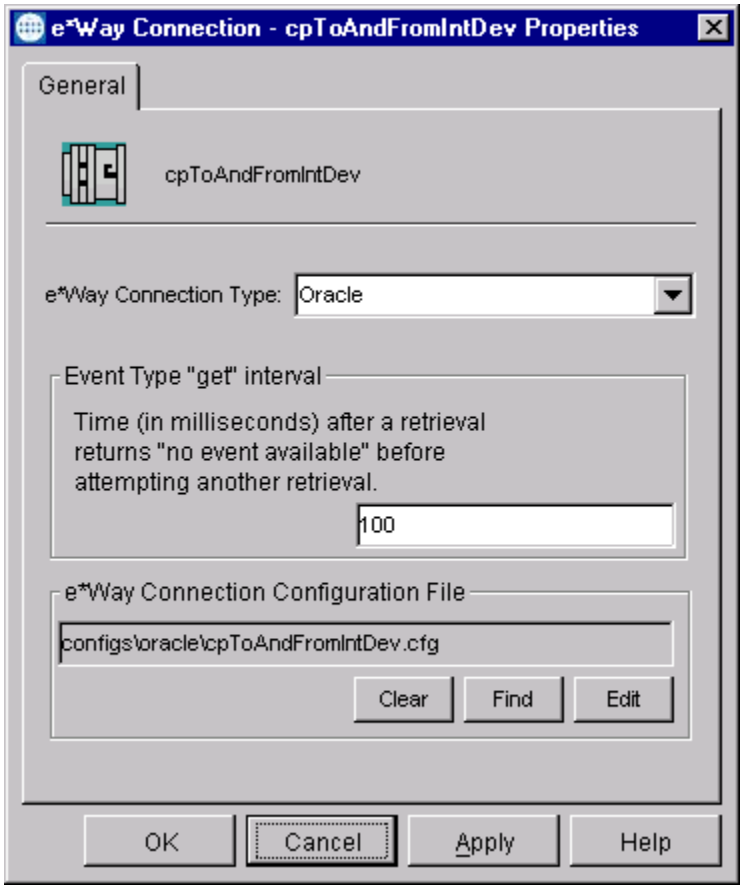


Figure 7-11: Oracle Connection Point Properties window

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:

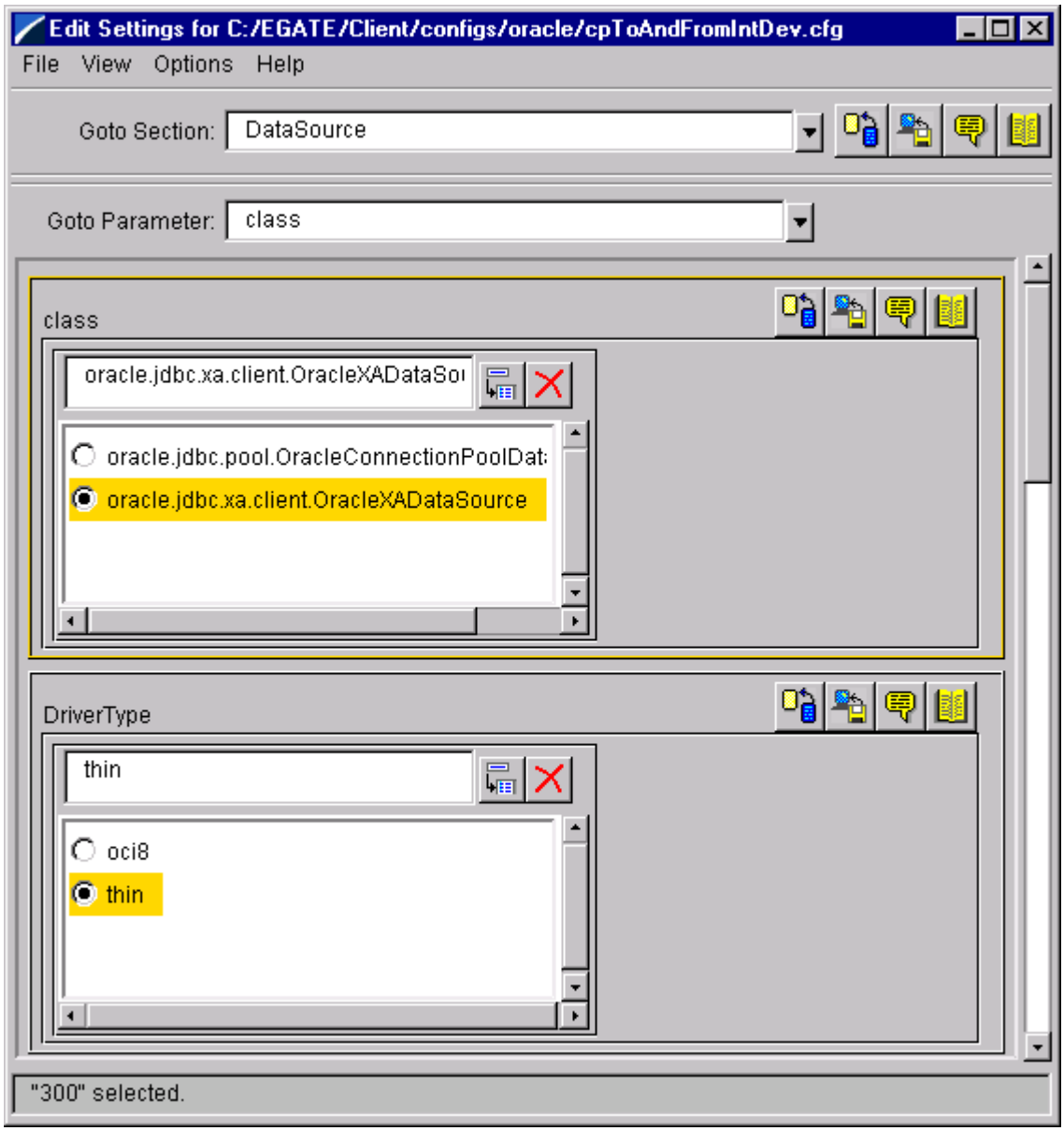


Figure 7-12: Oracle Connection Point Edit window

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.

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 RIP 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 (rib.properties). 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 10.0 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. This first one is the new event type used by the TAFR component to rout the message to the new destination. The second event type 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 10.0 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:

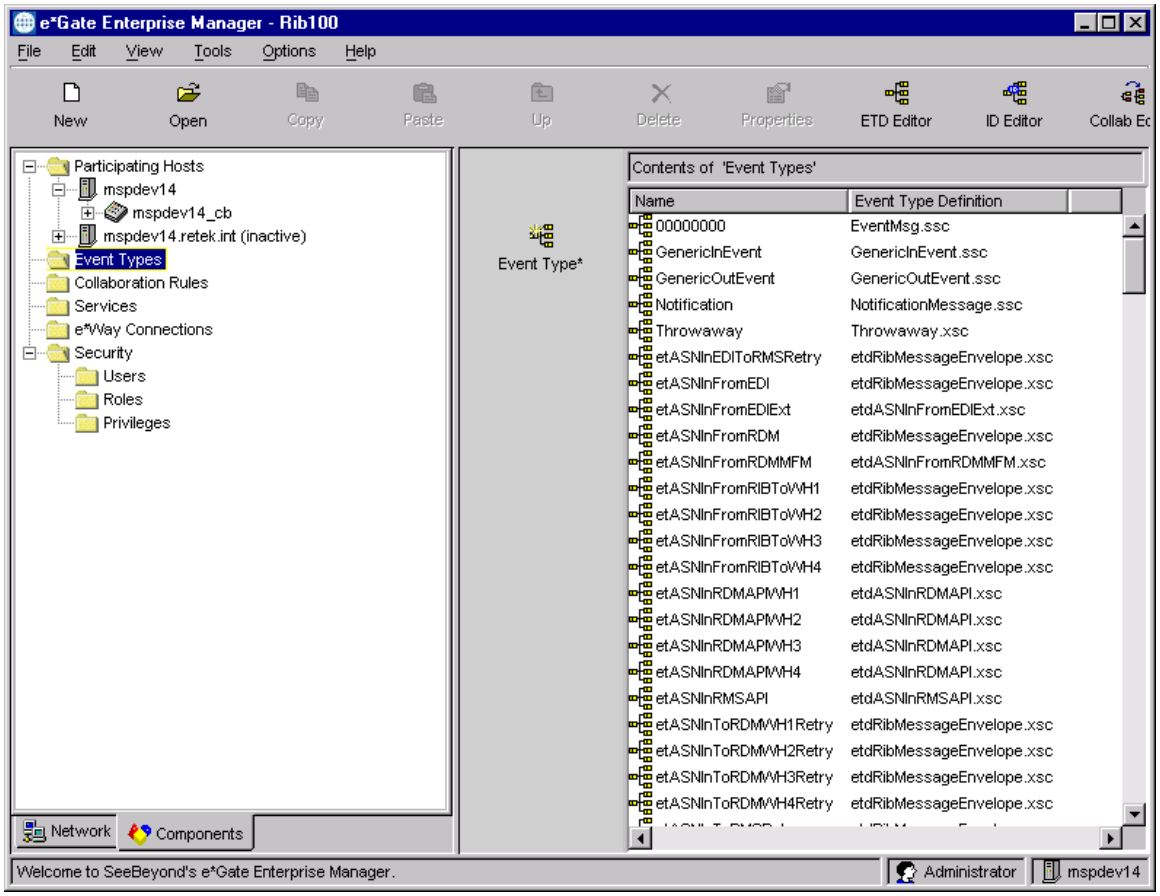


Figure 7-13: e*Gate Enterprise Manager with Event Types folder selected

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:

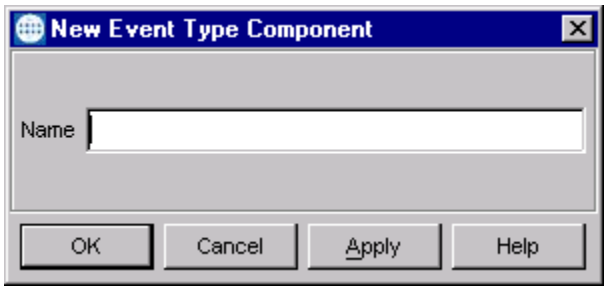


Figure 7-14: New Event Type window

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.

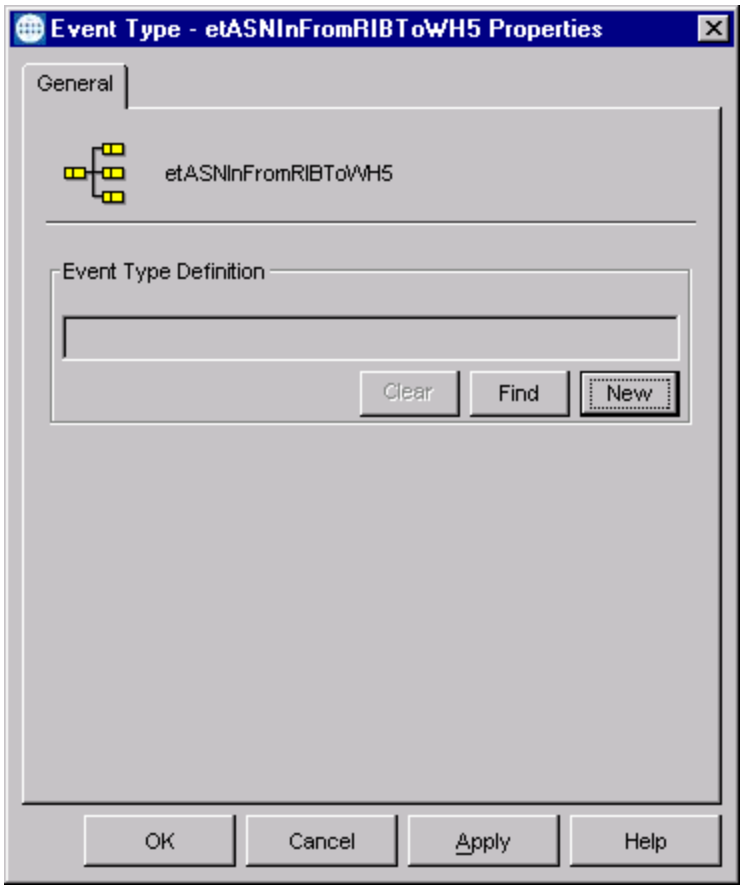


Figure 7-15: Event type properties window

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.

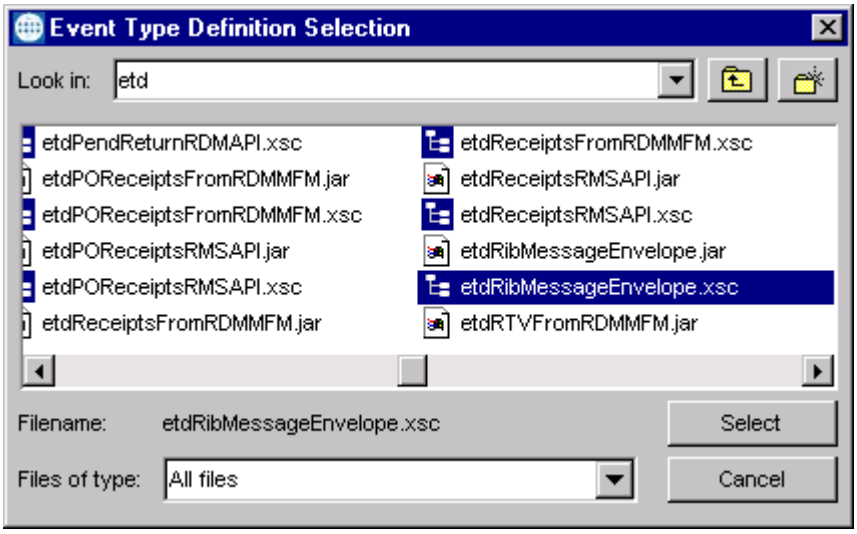


Figure 7-16: Choosing an Event Type Definition for the new Event Type

6 Select the etdRibMessageEnvelope.xsc file.

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

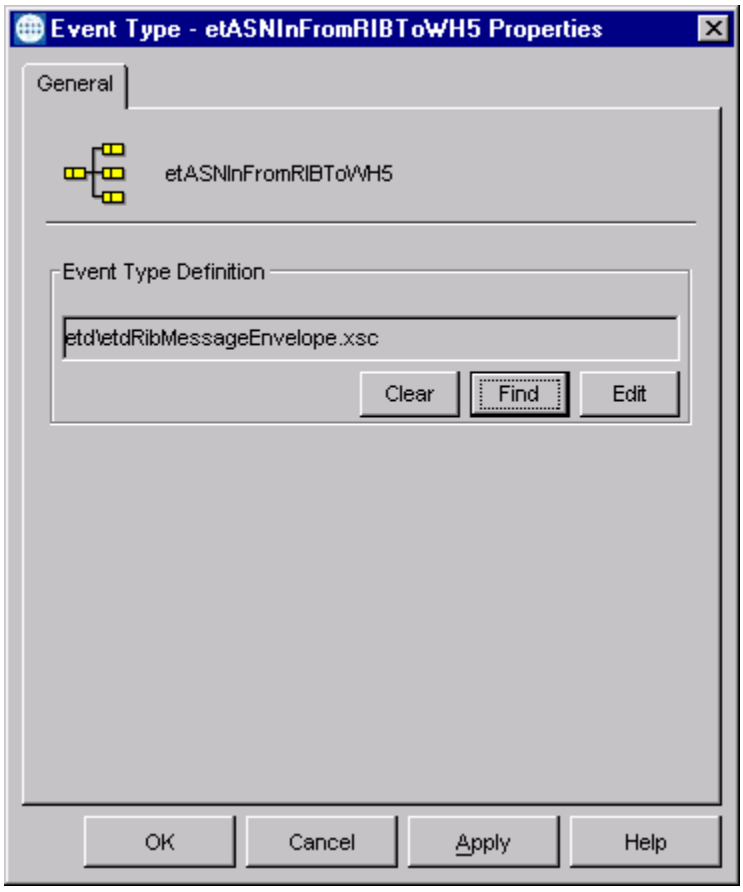


Figure 7-17: Updated Event Type Properties window

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.

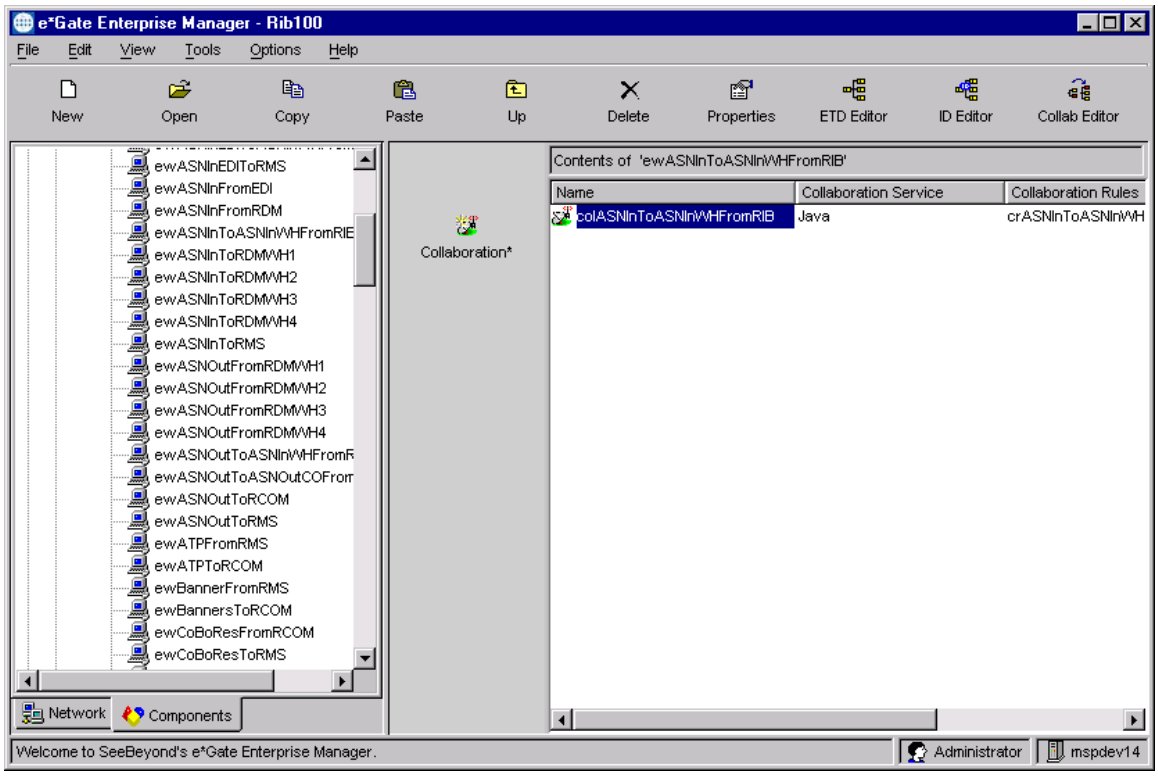


Figure 7-18: e*Gate Enterprise Manager with TAFR e*Way selected

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.

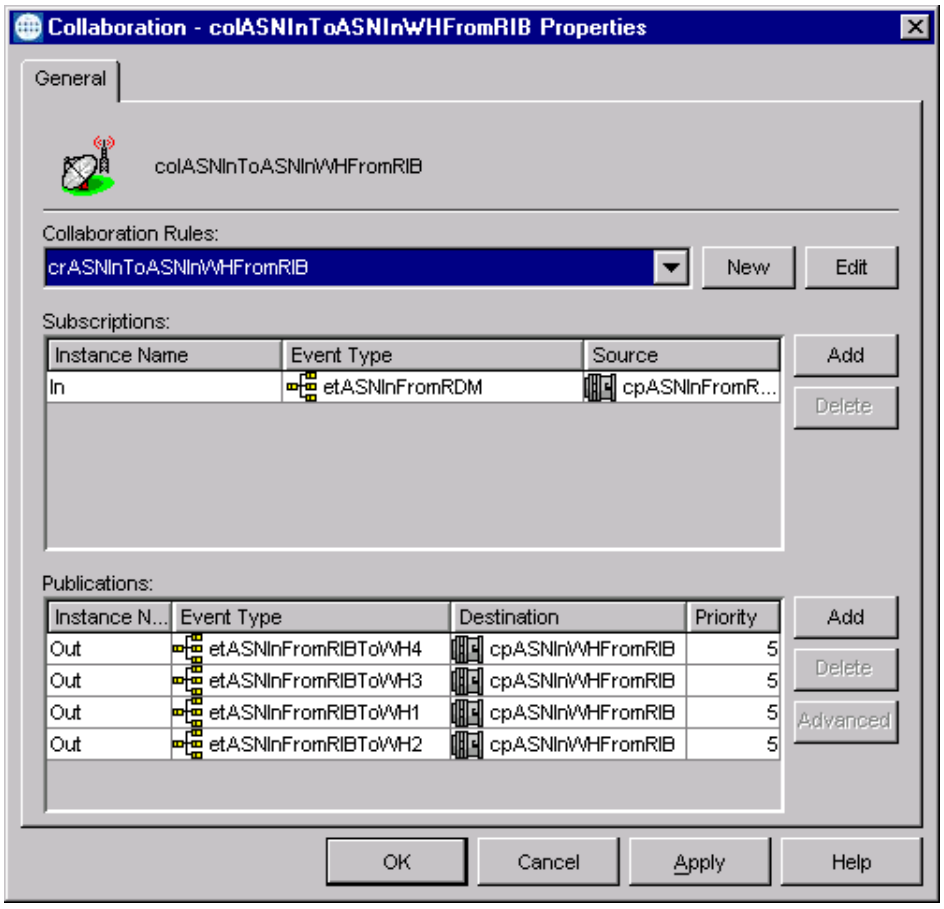


Figure 7-19: Collaboration Properties window

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.

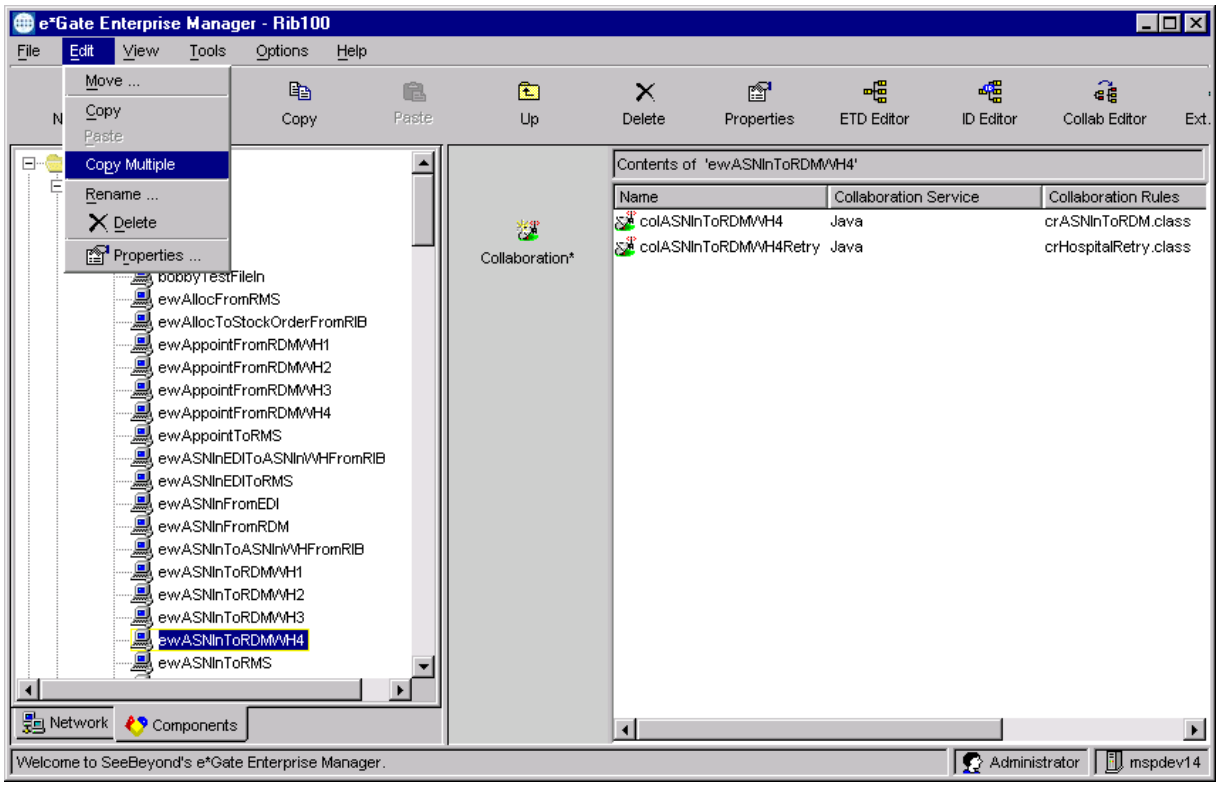


Figure 7-20: Copy Multiple edit option

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.

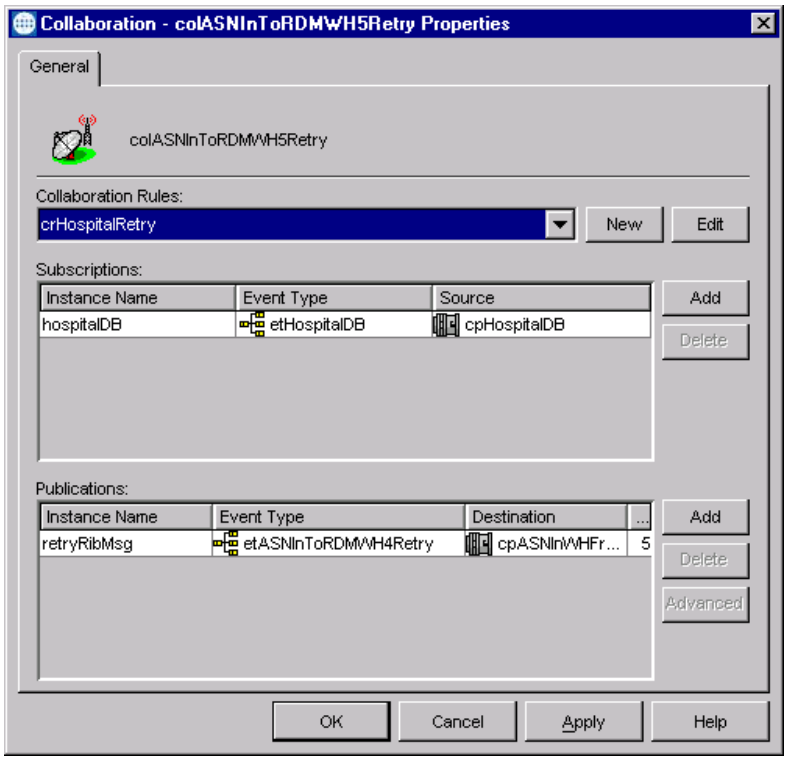


Figure 7-21: Collaboration Properties window for a Subscribing Application Retry collaboration.

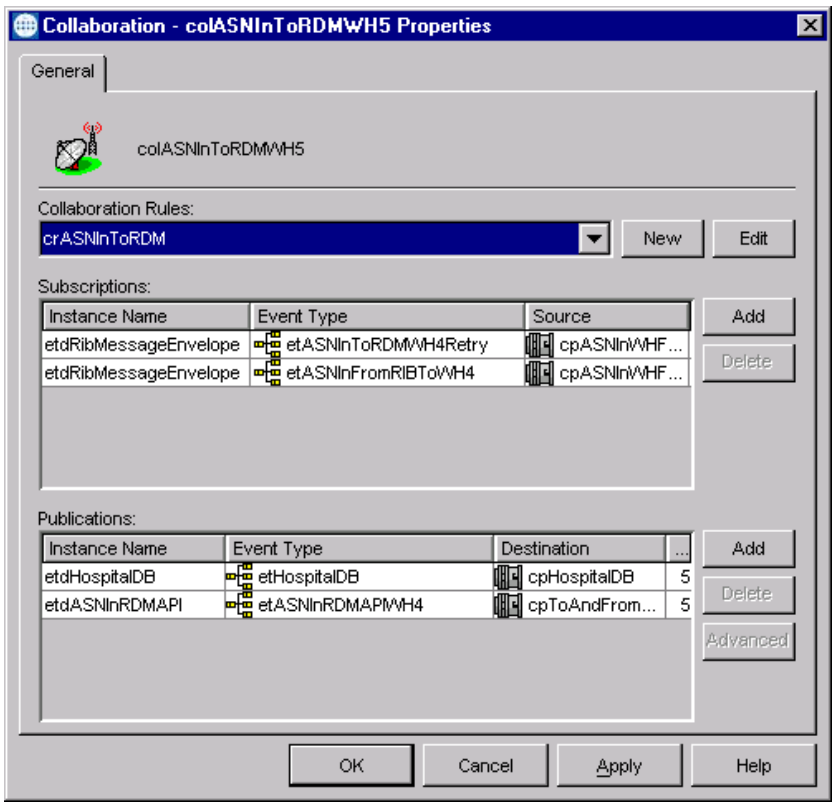


Figure7-22: Collaboration Properties window for the subscribing collaboration for a Subscribing Application adapter.

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.

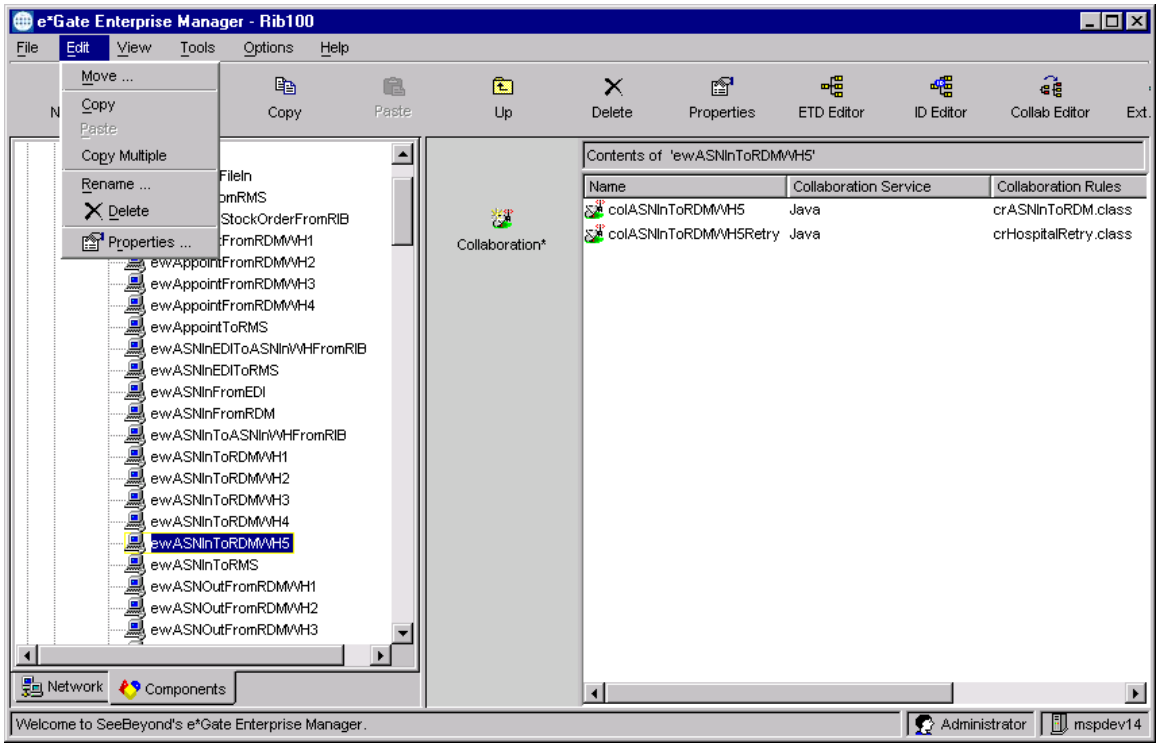


Figure 7-23: Edit drop-down menu.

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

Chapter 8 – Trouble-shooting problems

This section lists a general approach to trouble shooting problems.

If a problem persists, much information may 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 may 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 unless one accepts the risk of 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 <EGATE_HOME>/client/XALogs.
- **XA Not installed in Oracle:** An adapter may 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" will list running stccb processes. Use the "kill" command to bring down the extra stccb process

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.

No messages processed

Description: An adapter will not be 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 will ever be able to 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 will 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 will be 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, one should evaluate whether the transaction should be committed or rolled back administratively.

The following procedure will administratively commit 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 provisos 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". When examining the details of these alerts, it is seen that they are resulting from retrying 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 check 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, one can kill the process using the Unix "kill" command or, for Microsoft Windows platforms, the task manager.

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

Hospital Admin Command Line utility

There are two types of problems using the Hospital Command Line: Java class instantiation problems and Database connection problems.

Java Class Instantiation Problems

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

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

Database connection problems

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

Other possible connection problems include:

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