e*Way Intelligent Adapter for MQSeries User's Guide

Release 4.5.3

Java Version

SEEBEYOND™

Contents

Chapter 4

e*Way Connection Configuration (JMS)   22

Chapter 5

Implementation (JMS)   27

Chapter 9

Java Methods (ETD) 96

Appendix A

Appendix A (JMS) 131

Mapping Between JMS Standard Header Items and MQSeries Header Fields 131

Index 134

Introduction

This chapter introduces you to SeeBeyond™ Technology Corporation's (SeeBeyond™) Java-enabled e*Way Intelligent Adapter for the MQSeries. It includes an overview of this manual and a list of system requirements for installation.

1.1 Overview

MQSeries from IBM is a client-server message broker supporting an open API (application programming interface), available on a variety of operating systems, including AIX, Solaris, HP-UX, and Windows NT and 2000. MQSeries is "middleware" that provides commercial messaging and queuing services. Messaging enables programs to communicate with each other via messages instead of direct connection. Placing these messages in queues for temporary storage frees up programs to continue to work independently. This process also allows communication across a network of dissimilar components, processors, operating systems, and protocols.

The Java-enabled MQSeries e*Way allows the e*Gate system to exchange data with IBM's MQSeries version 5.2. The MQSeries e*Way applies business logic within Collaboration Rules to perform any of e*Gate's range of data identification, manipulation and transformation operations. Messages are tailored to meet the communication requirements of specific applications or protocols. Intelligent Queues (IQs) provide non-volatile storage for data within the e*Gate system allowing applications to run independently of one another at different speeds or times. Applications can freely send messages to a queue and get messages from a queue any time.

The MQSeries e*Way is configurable and transparently integrates existing systems and databases to IBM MQSeries through e*Gate. This document explains how to install and configure the Java-enabled MQSeries e*Way.

1.1.1 MQSeries e*Way JMS and ETD

The Java-enabled MQSeries e*Way is equipped for two different configuration modes, JMS-based and ETD-based. General implementation directions for each is provided in this document. Each provides advantages for different applications.

- The JMS-based MQSeries e*Way schema uses the Java Messaging System (JMS) e*Way connection and offers easy setup and high performance when connecting to a single queue.

- The ETD-based MQSeries e*Way schema relies on a fixed Event Type Definition (ETD) designed to expose various essential portions of the MQSeries Java API, providing a wide range of available methods and properties, as well as access to all message attributes. The ETD-based schema allows the e*Way to connect and switch between multiple queue managers and their queues.

Sections of this user guide that relate specifically to JMS-based or ETD-based MQSeries e*Way configuration and setup are marked with (JMS) or (ETD) in the chapter title.

1.1.2 Intended Reader

The reader of this guide is presumed to be a developer or system administrator with responsibility for maintaining the e*Gate system; to have a working knowledge of Windows or UNIX operations and administration; to be familiar with MQSeries, Java, and Windows-style GUI operations.

1.2 Supported Operating Systems

The MQSeries e*Way is available on the following operating systems:

- Windows 2000, Windows 2000 SP1, and Windows 2000 SP2
- Windows NT 4.0 SP6a
- Solaris 2.6, 7, and 8 (See note below regarding Solaris 2.6)
- AIX 4.3.3 and 5.1
- HP-UX 11.0 and HP-UX 11i (See note below regarding HP-UX 11)
- OS/390 V2R10 (Java ETD-based implementation only)
- Japanese Windows 2000, Windows 2000 SP1, and Windows 2000 SP2
- Japanese Windows NT 4.0 SP6a
- Japanese Solaris 2.6, 7, and 8
- Japanese HP-UX 11.0
- Traditional Chinese Windows 2000, Windows 2000 SP1, and Windows 2000 SP2 (Java only)
- Traditional Chinese Windows NT 4.0 SP6a (Java only)
- Traditional Chinese Solaris 8 (Java only)

Important: **The SupportPac MA88 patch is required for all supported platforms.** *See* **External System Requirements** *on page 10 for more information.*

Note: *Solaris 2.6 requires the installation of patches 105210-13 and 105568-10 available from Sun Microsystems at:* **http://sunsolve.sun.com/pub-cgi/show.pl?target=patches/patch-access**

> *Note:* *HP-UX Java binding support is only available for HP-UX 11 systems running the POSIX draft 10 threaded version of MQSeries. The HP-UX Developers kit for Java 1.1.7, Release C.01.17.01 or above is also required.*

1.3 System Requirements

To use the MQSeries e*Way, you need the following:

- An e*Gate Participating Host, version 4.5.1 or later.

- A TCP/IP network connection.

- Additional disk space for e*Way executable, configuration, library, and script files. The disk space is required on both the Participating and the Registry Host. Additional disk space is required to process and queue the data that this e*Way processes; the amount necessary varies based on the type and size of the data being processed, and any external applications performing the processing.

1.3.1 External System Requirements

The Java-enabled MQSeries e*Way requires the following installed on the participating host:

- IBM MQSeries version 5.2 (See note below).

> *Note:* *Install either the MQSeries server or the MQSeries client code. If the MQSeries server is installed on the participating host, the e*Way will connect to the server in bindings mode. If the MQSeries client code is installed on the participating host, the e*Way will connect to the server in client mode. A client mode connection can not participate in XA transactions.*

Install the following after installing IBM MQSeries version 5.2:

- IBM MQSeries classes for Java 5.2.0.

- Classes for Java Message Service 5.2.0.0

> *Important:* *The SupportPac MA88 patch is required for all supported platforms.*

The **MA88 SupportPac** download and installation information can be found at:
http://www-4.ibm.com/software/ts/mqseries/txppacs/ma88.html

The MA88 patch includes updates to several jar files and DLL's/ shared libraries. Most notably are **com.ibm.mq.jar**, **mqjbnd02.dll** and **mqxai01.dll**. It is important that the patch overwrites the existing versions of these files if they are present on your machine. Alternatively, if they do not overwrite the existing versions, it is important that the new versions of these files exist on your classpath and path before the old versions.

OS/390 Configuration Requirements for MQSeries

The Java-enabled MQSeries e*Way requires the following when installed on an OS/390 system. For additional installation notes for OS/390 V2R10, see the **e*Gate Integrator Installation Guide**.

The summarized requirements are as follows:

- The following DLL and Links must be copied from the MA88 mqm/java/lib directory to the e*Gate client/bin directory:

```
libwmqjbatch.so
libwmqjbatch_g.so -> libwmqjbatch.so
libwmqjbind.so
libwmqjbind_g.so -> libwmqjbind.so
libwmqjcics.so
libwmqjcics_g.so -> libwmqjcics.so
libwmqjrrs.so
libwmqjrrs_g.so -> libwmqjrrs.so
```

- The STEPLIB environment variable must be set prior to starting the MQ Series e*Way. It must point to the MQM.SCSQAUTH and MQM.SCSQANLE load libraries, as follows:

```
export STEPLIB=MQM.SCSQAUTH:MQM.SCSQANLE
```

Please note that on some versions of OS/390 the high level qualifier of these datasets are CSQ (not MQM).

- The OS/390 version of **com.ibm.mq.jar** from the MA88 Support Pac must be committed to the sandbox and promoted to runtime prior to running the MQSeries e*Way on OS/390. The MQ Series sample schema expects it to be located in the following directory:

```
../ThirdParty/mqseries/classes/
```

- The absolute path for the JNI DLL must be setup to point to the OS/390 JVM Shared Object for all Java e*Ways running on OS/390. For example:

```
/usr/lpp/java/J1.3/bin/classic/libjvm.so
```

Installation

This chapter explains how to install the Java-enabled MQSeries e*Way.

> **Windows NT and Windows 2000** on page 12
> **UNIX** on page 14
> **OS/390 V2R10** on page 15
> **Files/Directories Created by the Installation** on page 15

2.1 Windows NT and Windows 2000

2.1.1 Pre-installation

1 Quit all Windows programs before running the setup program, including any anti-virus applications.

2 You must have Administrator privileges to install this e*Way.

Important: ***The SupportPac MA88 patch is required for all supported platforms.*** *(See* **External System Requirements** *on page 10 for more information.)*

2.1.2 Installation Procedure

To install the MQSeries e*Way on a Windows system

1 Log in as Administrator on the work station to which you want to install the e*Way.

2 Insert the e*Way installation CD-ROM into the CD-ROM drive.

If Autorun is enabled, the setup program automatically starts. Otherwise:

  ◆ On the task bar, click the **Start** button, then click **Run**.

  ◆ In the **Open** field, type **D:\setup\setup.exe** where **D:** is your CD-ROM drive.

3 The InstallShield setup application launches. Follow the installation instructions until you come to the **Please choose the product to install** dialog box.

4 Select **e*Gate Integrator**, then click **Next**.

5 Follow the on-screen instructions until you come to the second **Please choose the product to install** dialog box.

6 Clear the check boxes for all selections except **Add-ons**, and then click **Next**.

7 Follow the on-screen instructions until you come to the **Select Components** dialog box.

8 Highlight (but do not check) **e*Ways**, and then click the **Change** button. The **Select Sub-components** dialog box appears.

9 Select **MQSeries e*Way** as shown in figure 1. Click the **Continue** button to return to the **Select Components** dialog box, then click **Next**.

Figure 1   Event Type Definition Wizard
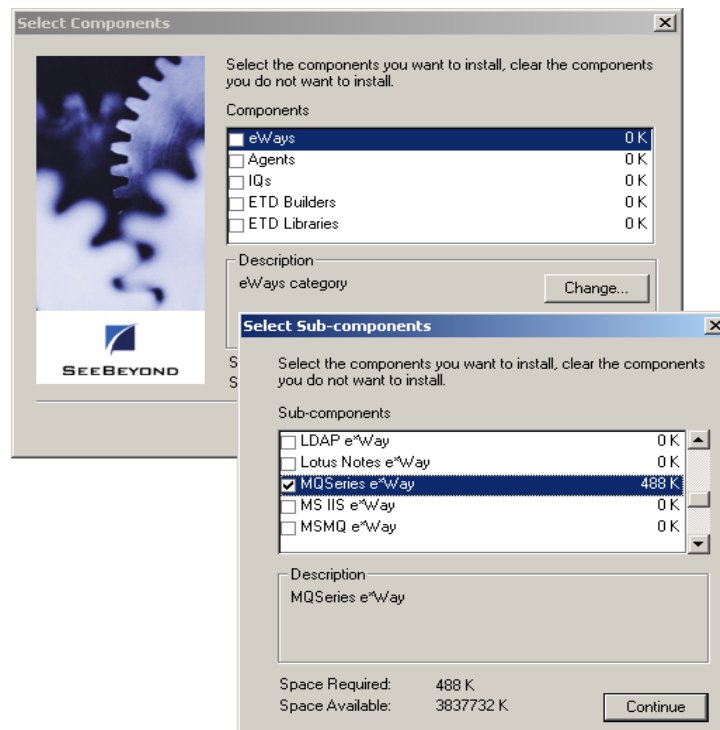
10 Follow the rest of the on-screen instructions to install the MQSeries e*Way. For details of e*Gate installation, refer to the *e*Gate Integrator Installation Guide*. Be sure to install the e*Way files in the suggested client installation directory. The installation utility detects and suggests the appropriate installation directory. Unless you are directed to do so by SeeBeyond support personnel, do not change the suggested installation directory setting.

Important: ***The SupportPac MA88 patch is required for all supported platforms.*** *(See* **External System Requirements** *on page 10 for more information.)*

Note: *Once you have installed and configured this e*Way, you must incorporate it into a schema by defining and associating the appropriate Collaborations, Collaboration Rules, IQs, and Event Types before this e*Way can perform its intended functions. For more information on any of these procedures, please see the online Help system.*

*For more information about configuring e*Ways or how to use the e*Way editors, see the* ***e*Gate Integrator User's Guide***.

2.2 UNIX

2.2.1 Pre-installation

You do not require root privileges to install this e*Way. Log in under the user name that you wish to own the e*Way files. Be sure that this user has sufficient privilege to create files in the e*Gate directory tree.

Important: **The SupportPac MA88 patch is required for all supported platforms.** *(See* **External System Requirements** *on page 10 for more information.)*

2.2.2 Installation Procedure

To install the MQSeries e*Way on a UNIX system

1 Log in on the workstation containing the CD-ROM drive, and insert the CD-ROM into the drive.

2 If necessary, mount the CD-ROM drive.

3 At the shell prompt, type:

cd /cdrom/setup

4 Start the installation script by typing:

setup.sh

5 A menu of options will appear. Select the **Install e*Ways** option. Then, follow any additional on-screen directions to install the **MQSeries e*Way**.

Note: *Be sure to install all files in the suggested* **client** *installation directory. The installation utility detects and suggests the appropriate installation directory. Unless you are directed to do so by SeeBeyond support personnel, do not change the suggested installation directory setting.*

6 After installation is complete, exit the installation utility and launch the Enterprise Manager.

Note: *Once you have installed and configured this e*Way, you must incorporate it into a schema by defining and associating the appropriate Collaborations, Collaboration Rules, Intelligent Queues (IQs), and Event Types before this e*Way can perform its intended functions. For more information about any of these procedures, please see the online Help system.*

*For more information about configuring e*Ways or how to use the e*Way editors, see the* **e*Gate Integrator User's Guide***.*

2.3 OS/390 V2R10

e*Way installation notes for OS/390 V2R10 can be found in the **e*Gate Integrator Installation Guide**.

2.4 Files/Directories Created by the Installation

The MQSeries e*Way installation process will install the files shown in Table 1 below within the e*Gate **client** directory tree. Files will be installed within the **egate\client** tree on the Participating Host and committed to the **default** schema on the Registry Host.

Table 1   Files Created by the Installation

Install Directory	Files
configs\mqseries	mqseries.def
etd\mqseriesetd\	MQSeriesETD.jar MQSeriesETD.xsc
configs\mqseriesetd	MQSeriesETD.def
etd\	mqseriesetd.ctl
ThirdParty\sun\	jta.jar

Multi-Mode e*Way Configuration

A Multi-Mode e*Way is a multi-threaded component used to route and transform data within e*Gate. Unlike traditional e*Ways, Multi-Mode e*Ways can use multiple simultaneous e*Way Connections to communicate with several external systems, as well as IQs or JMS IQ Managers. This chapter describes how to configure the Multi-Mode e*Way for the Java-enabled MQSeries e*Way.

3.1 Multi-Mode e*Way

Multi-Mode e*Way properties are set using the Enterprise Manager.

To create and configure a new Multi-Mode e*Way

1 Select the Navigator's Components tab.

2 Open the host on which you want to create the e*Way.

3 On the Palette, click on the icon to create a new **e*Way**.

4 Enter the name of the new e*Way, then click **OK**.

5 Select the new component, then right-click and select **Properties**. The e*Way Properties dialog box opens.

6 The **Executable File** field defaults to **stceway.exe**. (stceway.exe is located in the "bin\" directory).

7 Under the *Configuration File* field, click on the **New** button. When the Settings page opens, set the configuration parameters for this configuration file.

8 After selecting the desired parameters, save the configuration file. Close the **.cfg** file and select **OK** to close the e*Way Properties Window.

The Multi-Mode e*Way configuration parameters are organized in the JVM Settings section.

3.1.1 JVM Settings

The JVM Settings control the basic Java Virtual Machine configuration. Parameters are organized into the following sections.

- **JNI DLL Absolute Pathname** on page 17
- **CLASSPATH Prepend** on page 18

- **CLASSPATH Override** on page 18
- **CLASSPATH Append From Environment Variable** on page 18
- **Initial Heap Size** on page 19
- **Maximum Heap Size** on page 19
- **Maximum Stack Size for Native Threads** on page 19
- **Maximum Stack Size for JVM Threads** on page 19
- **Class Garbage Collection** on page 19
- **Garbage Collection Activity Reporting** on page 20
- **Asynchronous Garbage Collection** on page 20
- **Report JVM Info and all Class Loads** on page 20
- **Disable JIT** on page 20
- **Remote Debugging Port Number** on page 20
- **Suspend Option for Debugging** on page 21

JNI DLL Absolute Pathname

Description

Specifies the absolute pathname to where the JNI DLL installed by the *Java 2 SDK 1.3* is located on the Participating Host (for example, C:\eGate\client\bin\Jre or C:\jdk\jre\bin\server). This parameter is **mandatory**.

Required Values

A valid pathname.

Additional Information

The JNI dll name varies on different operating systems.

OS	Java 2 JNI DLL Name
Windows	jvm.dll
Solaris	libjvm.so
OS/390	libjvm.so
HP-UX	libjvm.sl
AIX 4.3.3	libjvm.a

The value assigned can contain a reference to an environment variable by enclosing the variable name within a pair of % symbols. For example:

```
%MY_JNIDLL%
```

Such variables can be used when multiple Participating Hosts are used on different platforms.

To ensure that the JNI DLL loads successfully, the Dynamic Load Library search path environment variable must be set appropriately to include all the directories under the Java 2 SDK (or JDK) installation directory that contain shared libraries (UNIX) or DLLs (NT).

CLASSPATH Prepend

Description

Specifies the paths to be prepended to the CLASSPATH environment variable for the Java VM.

Required Values

An absolute path or an environmental variable.

Additional Information

If left unset, no paths will be prepended to the CLASSPATH environment variable unless CLASSPATH Append From Environment Variable is set to yes, in which case the classpath will be appended to the variable.

Existing environment variables may be referenced in this parameter by enclosing the variable name in a pair of % signs. For example:

```
%MY_PRECLASSPATH%
```

CLASSPATH Override

Description

Specifies the complete CLASSPATH variable to be used by the Java VM. If left unset, an appropriate CLASSPATH environment variable (consisting of required e*Gate components concatenated with the system version of CLASSPATH) will be set.

*Note: All necessary .jar files and .zip files needed by both e*Gate and the Java VM must be included. It is advised that the **CLASSPATH Prepend** parameter be used.*

Required Values

An absolute path or an environmental variable.

Additional Information

Existing environment variables may be referenced in this parameter by enclosing the variable name in a pair of % signs. For example:

```
%MY_CLASSPATH%
```

CLASSPATH Append From Environment Variable

Description

Specifies the classpath append from the environmental variable.

Required Values

YES or **NO**. Enter Yes to append.

Initial Heap Size

Description

Specifies the value for the initial heap size in bytes. If set to 0 (zero), the preferred value for the initial heap size of the Java VM will be used.

Required Values

An integer between 0 and 2147483647.

Maximum Heap Size

Description

Specifies the value of the maximum heap size in bytes. If set to 0 (zero), the preferred value for the maximum heap size of the Java VM will be used.

Required Values

An integer between 0 and 2147483647.

Maximum Stack Size for Native Threads

Description

Specifies the value of the maximum stack size in bytes for native threads. If set to 0 (zero), the default value will be used.

Required Values

An integer between 0 and 2147483647.

Maximum Stack Size for JVM Threads

Description

Specifies the value of the maximum stack size in bytes for JVM threads. If set to 0 (zero), the preferred value for the maximum heap size of the Java VM will be used.

Required Values

An integer between 0 and 2147483647.

Class Garbage Collection

Description

Specifies whether the Class Garbage Collection will be done automatically by the Java VM. Reserved for future use. Do not change from default value.

Required Values

YES or **NO**.

Garbage Collection Activity Reporting

Description

Specifies whether garbage collection activity will be reported for debugging purposes. Reserved for future use. Do not change from default value.

Required Values

YES or **NO**.

Asynchronous Garbage Collection

Description

Specifies whether asynchronous garbage collection activity will be reported for debugging purposes.

Required Values

YES or **NO**.

Report JVM Info and all Class Loads

Description

Specifies whether the JVM information and all class loads will be reported for debugging purposes. Reserved for future use. Do not change from default value.

Required Values

YES or **NO**.

Disable JIT

Description

Specifies whether the Just-In-Time (JIT) compiler will be disabled.

Required Values

YES or **NO**.

Note: *This parameter is not supported for Java Release 1.*

Remote Debugging Port Number

Description

Specifies the port number for remote debugging of JVM.

Required Values

An integer ranging from 2000 and 65536.

Suspend Option for Debugging

Description

Specifies whether to suspend the debugging of JVM option.

Required Values

YES or **NO**.

e*Way Connection Configuration (JMS)

This chapter describes how to configure the Java-enabled MQSeries e*Way Connection using the MQSeries JMS connection type.

4.1 Configuring e*Way Connections

e*Way Connections are set using the Enterprise Manager.

To create and configure e*Way Connections

1 In the Enterprise Manager's **Component** editor, select the **e*Way Connections** folder.

2 On the palette, click on the **Create a New e*Way Connection** button.

3 The **New e*Way Connection Component** dialog box opens, enter a name for the **e*Way Connection**. For the purposes of the sample implementation enter **MQ_conn1** as the name.

4 Double-click on the new **e*Way Connection**. The **e*Way Connection Properties** dialog box opens.

5 From the **e*Way Connection Type** drop-down box, select **MQSeries JMS**.

6 Enter the **Event Type "get" interval** in the dialog box provided. 10000 milliseconds is the configured default. The "get" interval is the intervening period at which, when subscribed to, the e*Way connection is polled.

7 From the **e*Way Connection Configuration File**, click **New** to create a new Configuration File for this e*Way Connection. (To use an existing file, click **Find**.)

The MQSeries e*Way Connection configuration parameters are organized into the following sections.

  ▪ **General Settings**
  ▪ **MQSeries**

4.1.1 General Settings

This section contains a set of top level parameters:

  ▪ **Connection Type**

- **Transaction Type**
- **Delivery Mode**
- **Maximum Number of Bytes to read**
- **Default Outgoing Message Type**
- **Factory Class Name**

Connection Type

Description

String-set. Specifies the JMS Messaging Model. Only Queue connection type is supported.

- Queue (point to point)

Required Values

The value is Queue. Queue is the configured default.

Transaction Type

Description

String-set. Specifies the Transaction Type. There are three transaction types.

- **Internal**: Provides protection for transactions sent internally between IBM MQSeries and e*Way queues. In the event of a system error, messages in transit are rolled back, restoring the message. When the send() method is called the transaction takes place at the end of the collaboration.

- **Non-Transactional**: Provides the highest level of performance, with the minimum level of message protection. No rollback is available during the send and receive period, causing the possible loss of data in the case of a system error. When the send() method is called the transaction is immediate.

- **XA-compliant**: (two-phase transactional behavior) Highest level of transaction protection, providing rollback for internal and XA compliant transactions. The transaction is also extended to other XA supported data exchange applications, such as Oracle, DB2, and MQSeries. When the send() method is called the transaction takes place at the end of the collaboration.

Required Values

A valid transaction type. One of three provided: Internal, Non-Transactional or XA-compliant. Internal is the configured default.

Note: *Consult the XA Processing chapter of the e*Gate User's Guide for information on XA use and restrictions.*

Delivery Mode

Description

String-set. Specifies the Message Delivery Mode. Marking the message as follows.

- **Non-Persistent**. Provides the highest performance. The message is cashed in memory during the transaction.

- **Persistent**. Provides the highest level of protection, ensuring that the message will be saved to a reliable persistent store by the Message Server before the publish() method returns.

This setting must match the setting in the IBM MQSeries queue manager.

Required Values

Non-Persistent or Persistent. Persistent is the configured default.

Maximum Number of Bytes to read

Description

Integer-set. Specifies the maximum number of bytes to read at a time from the received Bytes Message.

Required Values

An integer in the range of 1 to 104,857,600. The configured default is 8192.

Default Outgoing Message Type

Description

String-set. Specifies the Message Type to create during publish/send. The outgoing message type is published within the message header. This is only relevant to sending, providing information for the receiver.

Required Values

Bytes or Text. The configured default is Bytes.

Factory Class Name

Description

String-set. Specifies the Factory class to be used to connect to the JMS IQ Manager. This is advanced configuration to be utilized in future development, and should not be changed from the default.

Required Values

The valid factory class name. The configured default is com.stc.common.collabService.MQJMSFactory. Retain the default setting.

4.1.2 MQSeries

This section contains a set of top level parameters:

- **Queue Manager Name**
- **Transport Type**
- **Host Name**
- **Port Number**
- **Channel**

Queue Manager Name

Description

String-set. Specifies the name of the IBM MQSeries Queue Manager.

Required Values

Enter the name of the IBM MQSeries Queue Manager.

Transport Type

Description

String-set. Specifies the Transport Type:

- Client
- Binding

JMS can communicate with MQSeries using either the client or bindings transports. Use of the Java bindings requires the JMS application and the MQSeries queue manager to be located on the same machine. The client permits the queue manager to be on a different machine to the application. Binding has a performance advantage but requires a local queue manager.

Required Values

Select Client or Binding. The configured default is Client

Host Name

Description

String-set. Specifies the name of the host on which the queue manager resides. This option is only relevant with transport type 'Client' and will be ignored for transport type 'Bindings' as for 'Bindings' the queue manager has to be on the same host.

Required Values

Enter the name of the queue manager host.

Port Number

Description

Integer-set. Specifies the number of the port to connect to. This option is only relevant with a transport type 'Client' and will be ignored for transport type 'Bindings'. If this option is left empty the default port will be used.

Required Values

Enter the port number, in the range of 1000 and 65536. The configured default is 1414.

Channel

Description

String-set. Specifies the name of the channel being used. This option is only relevant with transport type 'Client' and will be ignored for transport type 'Bindings'. If no channel is specified the default channel will be used.

Required Values

The valid name of the channel.

Implementation (JMS)

This chapter contains basic information for implementing the Java-enabled MQSeries e*Way in the MQSeries JMS mode, in a production environment. Examples are given for creating and configuring the necessary components to implement the sample MQSeries schema included on the CD-ROM. For more information on creating and configuring e*Way components see the *e*Gate Integrator User's Guide*.

5.1 MQSeries e*Way Implementation Overview

The Java enabled MQSeries e*Way is an *application specific* e*Way which allows e*Gate to connect with IBM's MQSeries applications. When the MQSeries e*Way is installed along with the e*Gate Integrator, schema's can be created and configured using the e*Gate Enterprise Manager. A schema is an organization scheme that contains the parameters of all the components that control, route, and transform data as it moves through e*Gate in a predefined system configuration. To create an e*Gate schema for MQSeries you must do the following:

- **Install IBM's MQSeries Server 5.2 and MQSeries Queue Manager:** The MQSeries Server 5.2 and MQSeries Queue Manager are installed on the localhost. Also install **Java Classes for MQSeries 5.2** and the **MA88 SupportPac from IBM.** (See **External System Requirements** on page 10.)

- **Install the MQSeries e*Way:** The MQSeries e*Way is installed as an Add-on to the E*Gate Integrator. For directions on installing the MQSeries e*Way from CD-ROM on your specific operating system. (See **Installation** on page 12.)

- **Create e*Ways:** e*Ways connect with external systems to poll or send data. They also transform and route data. Multi-Mode e*Ways are used to run Java Collaborations that utilize e*Way Connections to send and receive Events to and from multiple external systems.

- **Configure e*Way Connections:** An e*Way Connection is the encoding of access information for a particular external connection. The e*Way Connection configuration file contains the parameters necessary for communicating with IBM's MQSeries and specifying the MQSeries Queue Manager.

- **Create Event Types:** Each packet of data within e*Gate is referred to as an Event. Event Types are data labels that allow e*Gate to process and route specific Events differently. The Event Type specifies the MQSeries Queue (the Event Type must

have the same name as the IBM's MQSeries Queue). Data is not routed in e*Gate without an Event Type.

- **Create Intelligent Queues:** Intelligent Queues (IQs) provide non-volatile storage for data traveling through the e*Gate system. The IQ Manager oversees the activities of the individual storage locations. The exact behavior of each IQ is determined by the IQ Service configuration. The MQSeries e*Way uses the MQSeries IQ Service.

- **Create Collaboration Rules:** Collaboration Rules determine how input Event Types are modified to the format of specific output Event Types. A Collaboration Rule defines what type of data is received, how it will be transformed and what type of data will be published.

- **Create Collaborations:** A Collaboration is a message bus in e*Gate that specifies the name and source of the incoming Event Types, the Collaboration Rules that will be applied to the Event, and the name, destination and expiration date of the outgoing Event Types. A Collaboration designates the Subscriber, which "listens" for Events of a known type from a given source, and the Publisher, which distributes the transformed Event to a specified recipient.

- **Set the CLASSPATH Variable:** The Final Step in creating and configuring the MQSeries e*Way is to set the IBM MQSeries Java .jar files in the environment CLASSPATH variable.

5.2 MQSeries Sample Schema Components

A sample schema for MQSeries is available in the samples folder on the CD-ROM. In addition, the following pages explain how the components for the MQSeries sample schema were created. The Host and Control Broker are automatically created and configured during the e*Gate installation. The default name for each is the name of the host on which you are installing the e*Gate Enterprise Manager GUI. To complete the sample implementation of the Java-enabled MQSeries e*Way requires the following components:

- IBM MQSeries Server and the MQSeries Queue Manager.

- Java Classes for MQSeries 5.2.

- Install the MQSeries e*Way Add-on. Make sure that the Control Broker is activated

- In the e*Gate Enterprise Manager, define and configure the following as necessary:

  - Inbound e*Way using **stcewfile.exe** as the executable file.

  - Outbound e*Way using **stcewfile.exe** as the executable file.

  - The Multi-Mode e*Way component using **stceway.exe** as the executable file.

  - Event Type Definitions used to package the data to be exchanged with the external system.

  - Intelligent Queues (IQs) to provide non-volatile storage Events

  - Collaboration Rules to process Events.

- The e*Way Connection to be created as described in **Chapter 4**.

- Collaborations, to be associated with each e*Way component, to apply the required Collaboration Rules.

- The destination to which data will be published prior to being sent to the external system.

The following sections describe how to define and associate each of the above components. This sample implementation will demonstrate how the Java-enabled MQSeries e*Way intercepts, stores, manipulates, and manages data in association with IBM MQSeries.

Figure 2 The MQSeries (JMS) Sample Implementation
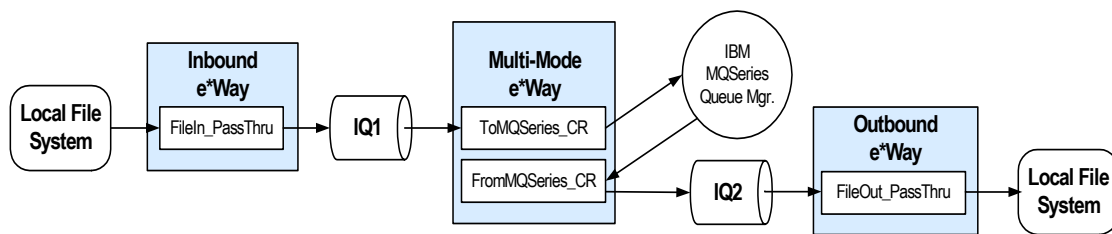
5.2.1 Step One: Creating the IBM MQSeries Queue

Step one in creating the MQSeries e*Way is to install and configure **IBM MQSeries** (See **External System Requirements** on page 10) and the **IBM MQSeries Queue Manager** on the localhost. Also install **Java Classes for MQSeries 5.2.** It is assumed that the reader is experienced in the use of IBM MQSeries Queue Manager. For more information on IBM MQSeries Queue Manager please see MQSeries Queue Related Commands, Chapter 9, in the e*Gate Integrator Intelligent Queue Services Reference Guide. For the sample implementation do the following:

1 Open IBM MQSeries Explorer.

2 Create a new Queue Manager named **Java_On**.

3 From the Java_On Queue Manager create a new queue named **Ev_1**.

Important: *The MQSeries Queue name and the Event Type name must be the same.*

IBM MQSeries Server and Queue Manager Limits and Settings

- When using MQSeries Queue Manager on UNIX, the user must be a member of the mqm group to create and start MQ Series Queue manager.

- It is essential that the MQSeries Administrator regularly monitor the number of messages in the queue. Message expiration settings should be set to allow for extended storage.

- MQseries is limited in the number of messages that can be sent before a commit is executed and the number of physical messages that can exist on the queue at any one time. This can result in exception errors when the upper limit for these numbers

is exceeded. Memory and performance of the specific server may also effect the results.

Publishing Messages with MQSeriesJMS to a non-JMS conversant e*Way

The JMS standard specifies a header which includes encoding and reply information. This header is prepended to any message published by the IBM JMS classes. Non-JMS subscribers (that is, those using a non JMS API to MQSeries, such as the IBM C API) will not be able to separate the JMS header from the body. To remedy this, the user is advised to suppress the publication of the JMS header, if publishing to non-JMS subscribers, using the following mechanism.

To send messages to a non-JMS MQ (Monk MQSeries) e*Way, call send( ) manually from within the collaboration rules containing the following URI:

"queue://<QMGR_NAME>/<QNAME>?targetClient=1"

For example:

"queue://EMEO2T/QR.EME01?targetClient=1"

If this is not done then a 200+ byte header is pre-appended to the payload and placed in the MQ queue and could easily throw off the non JMS conversant MQ reader.

For more information see the IBM Corp. manual MQSeries Using Java, Chapter 10, at: **http://www-4.ibm.com/software/ts/mqseries/library/manualsa/manuals/ crosslatest.html**

5.2.2 Step Two: Installing the MQSeries e*Way and Creating a New Schema

Step two is to install the MQSeries e*Way. For directions on installing the MQSeries e*Way on your specific operating system, see **Chapter 2, Installation, on page 12**.

Once the MQSeries e*Way is installed, a new schema must be created. While it is possible to use the default schema for the sample implementation, it is recommended that you create a separate schema for testing purposes. After you install the MQSeries e*Way, do the following:

1 Start the e*Gate Enterprise Manager GUI.

2 When the Enterprise Manager prompts you to log in, select the host that you specified during installation and enter your password.

3 You will then be prompted to select a schema. Click on **New**.

4 Enter a name for the new Schema. In this case, for the sample implementation, enter **MQSSample**, or any name as desired.

The e*Gate Enterprise Manager opens to your new schema. You are now ready to begin creating the necessary components for this schema.

5.2.3 Step Three–Creating and Configuring the e*Ways

Step three is to create the e*Ways. These are used as components for transporting and transforming data. They always interface with at least one external system, and Multi-

Mode e*Ways can use e*Way Connections to interface with many external systems. For the sample implementation three e*Ways are required.

- Inbound_eWay
- Outbound_eWay
- Multi-Mode_eWay

The following sections provide instructions for creating each e*Way.

Inbound e*Way

1 Select the Navigator's **Components** tab.

2 Open the host on which you want to create the e*Ways.

3 Select the **Control Broker** that will manage the new e*Ways.

4 On the palette, click the **Create a New e*Way** button.

5 Enter the name of the new e*Way. In this case, **ew_In.** Click **OK**.

6 Right-click **ew_In**, and select **Properties** to edit its properties.

7 When the e*Way Properties window opens, click on the **Find** button beneath the **Executable File** field and select **stcewfile.exe** as the executable file.

Figure 3 e*Way Sample Implementation
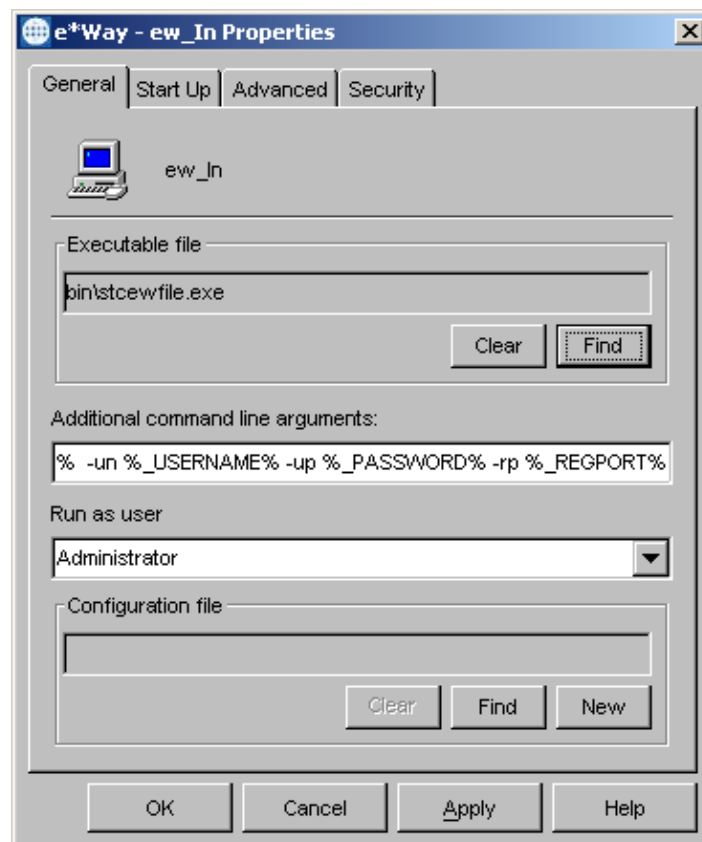
8   Under the **Configuration File** field, click on the **New** button. The **Edit Settings** dialog box opens. Set the following for this configuration file.

Table 2   Configuration Parameters for the Inbound e*Way

Parameter	Value
General Settings (unless otherwise stated, leave settings as default)	
AllowIncoming	Yes
AllowOutgoing	No
Outbound Settings	Default
Poller Inbound Settings	
PollDirectory	C:\Indata (input file folder)
InputFileMask	*.fin (input file extension)
PollMilliseconds	Default
Remove EOL	Default
MultipleRecordsPerFile	Yes
MaxBytesPerLine	Default
BytesPerLineIsFixed	Default
File Records Per eGate Event	Default
Performance Testing	Default

Note: *For information on configuring the specific parameters of the stcewfile e*Way see the* ***Standard e*Way Intelligent Adapter User's Guide****.*

9   After selecting the desired parameters, save the configuration file (**ew_In.cfg**) and Promote to Run Time. Close the **.cfg** file.

10   Use the Startup, Advanced, and Security tabs to modify the default settings for each e*Way you configure.

   A   Use the **Startup** tab to specify whether the e*Way starts automatically, or restarts after abnormal termination or due to scheduling and so forth.

   B   Use the **Advanced** tab to specify or view the activity and error logging levels as well as the Event threshold information.

   C   Use **Security** to view or set privilege assignments.

11   Select **OK** to close the **e*Way Properties** window.

Outbound e*Way

1   Select the Navigator's **Components** tab.

2   Open the host on which you want to create the e*Ways.

3   Select the **Control Broker** that will manage the new e*Ways.

4   On the palette, click the **Create a New e*Way** button.

5   Enter the name of the new e*Way, (in this case, **ew_Out**), then click **OK**.

6 Select **ew_Out**, then right-click and select Properties to edit its properties.

7 When the e*Way Properties window opens, click the **Find** button beneath the **Executable File** field, and select **stcewfile.exe** as the executable file.

8 Under the **Configuration File** field, click the **New** button. When the **Settings** page opens, set the following for this configuration file.

Table 3 Configuration Parameters for the Outbound e*Way

Parameter	Value
General Settings (unless otherwise stated, leave settings as default)	
AllowIncoming	No
AllowOutgoing	Yes
Outbound Settings	
OutputDirectory	C:\DATA
OutputFileName	output%d.dat
MultipleRecordsPerFile	No
MaxRecordsPerFile	10000
AddEOL	Yes
Poller Inbound Settings	Default
Performance Testing	Default

9 Save the **.cfg** file (**ew_Out.cfg**), and promote to run time.

10 Click **OK** to close **e*Way Properties** window.

Multi-Mode e*Way

1 Select the **Navigator's Components** tab.

2 Open the host on which you want to create the e*Way.

3 Select the **Control Broker** that will manage the new e*Way.

4 On the palette, click the **Create a New e*Way** button.

5 Enter the name of the new e*Way (in this case, **MQ_stceway**), then click **OK**.

6 Right-click the new e*Way and select **Properties** to edit its properties.

7 When the e*Way Properties window opens, click the **Find** button beneath the **Executable File** field, and select **stceway.exe** as the executable file.

8 To edit the JVM Settings, select **New** under Configuration file.

See **"Multi-Mode e*Way Configuration" on page 16** for details on the parameters associated with the Multi-Mode e*Way.

Table 4 Configuration Parameters for the MultiMode e*Way

Parameter	Value
JVM Settings (unless otherwise stated, leave settings as default)	

Table 4 Configuration Parameters for the MultiMode e*Way

Parameter	Value
JNI DLL absolute pathname	C:\eGate\client\bin\Jre\jvm.dll (or absolute path to proper JNI DLL)
CLASSPATH Append From Environmental Variable	Yes

9 Save the **.cfg** file (**MQ_stceway**).

10 Go to **File** and click **Promote to Run Time**.

11 In the **e*Way Properties** window, use the **Startup**, **Advanced**, and **Security** tabs to modify the default settings for each.

    D Use the **Startup** tab to specify whether the e*Way starts automatically, restarts after abnormal termination or due to scheduling, etc.

    E Use the **Advanced** tab to specify or view the activity and error logging levels, as well as the Event threshold information.

    F Use **Security** to view or set privilege assignments.

12 Click **OK** to close **e*Way Properties** window.

5.2.4 Step Four: Create the e*Way Connection (Includes Specifying the MQSeries Queue Manager)

Step four is to create and configure the e*Way Connection. The e*Way Connection configuration file contains the connection settings necessary for communicating with IBM MQSeries and specifying the MQSeries Queue Manager.

To create and configure a New e*Way Connection

1 Select the **e*Way Connection** folder on the **Components** tab of the e*Gate Navigator.

2 On the palette, click the **Create a New e*Way Connection** button.

3 Enter the name of the **e*Way Connection**, then click **OK.** (For the purpose of this sample, the e*Way Connection is defined as "**MQ_conn1**".)

4 Double-click the new **e*Way Connection** to edit its properties. The **e*Way Connection Properties** dialog box opens.

5 In the e*Way Connection Type field, select **MQSeries JMS** from the drop-down list box.

Figure 4 e*Way Connection Properties
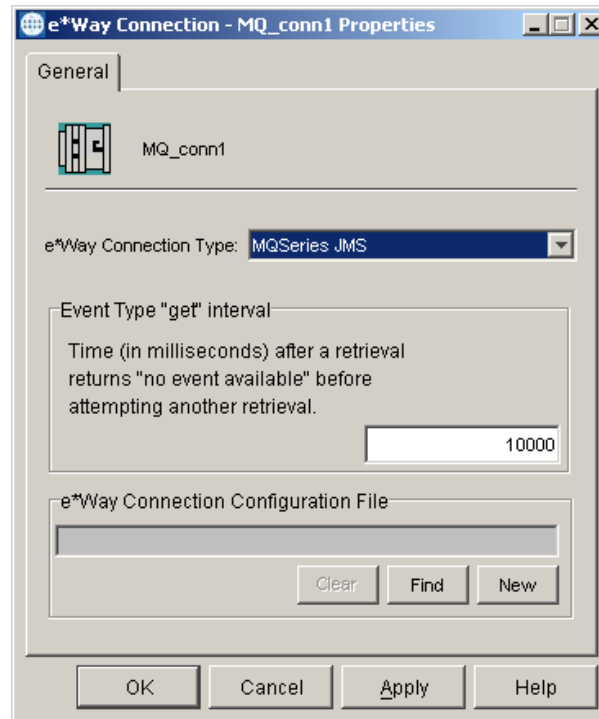
6 Enter the **Event Type "get" interval** in the dialog box provided. 10000 milliseconds is the configured default. The "get" interval is the intervening period at which, when subscribed to, the e*Way connection is polled.

7 Under e*Way Connection Configuration File, click the **New** button.

8 The e*Way Connection editor opens, select the necessary parameters. For more information on the MQSeries e*Way Connection Type parameters, see **"Configuring e*Way Connections" on page 22**.

9 Save the **MQ_conn1.cfg** file.

10 From the File menu select **Promote to Run Time** to promote the file to the e*Way's run time environment.

5.2.5 Step Five: Creating Event Types (Also Specifies MQSeries Queue)

Step five is to create the Event Type. This also specifies the MQSeries queue (the Event Type must have the same name as the IBM MQSeries queue). An Event Type is a class of Events with a common data structure. The e*Gate system packages data within Events and categorizes them into Event Types. What these Events have in common

defines the Event Type and comprises the ETD. The following procedures show how to create an **ETD** (Event Type Definition) using the Standard ETD Wizard.

1 Highlight the **Event Types** folder on the **Components** tab of the e*Gate Navigator.

2 On the palette, click the **Create a New Event Type** button.

3 Enter the name of the **Event**, then click **OK.** For the purpose of this sample the first Event Type is defined as **Ev_1**.

Important: ***The Event Type must have the same name as the IBM MQSeries Queue.***

4 Double-click the new **Event Type** to edit its properties. The **Event Type Properties** dialog box opens.

5 Click the **New** button. The ETD Editor opens.

6 Select **New** from the File menu. The New Event Type Definition window opens.

Figure 5   Event Type Definition Wizards
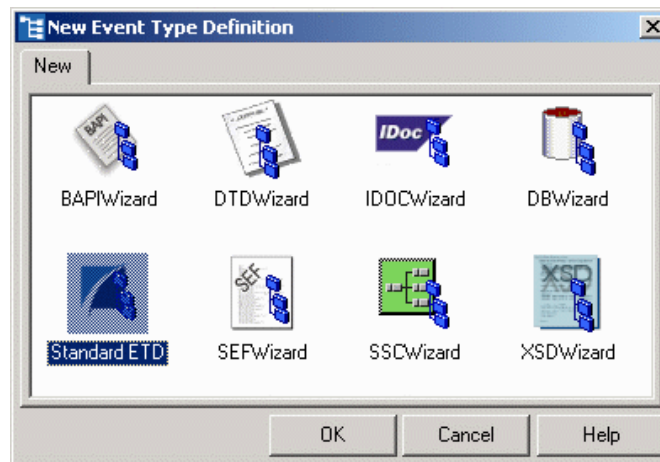
7 Select the appropriate wizard. (For this Event Type, select the **Standard ETD** wizard.)

8 Enter the Root Node Name (for this case, "**Record**").

9 Enter a package name where the ETD Editor can place all the generated Java classes associated with the created ETD. (For this sample, use **com.stc.eway.mqseries** as the package name.)

10 Right click **Record** in the Event Type Definition pane, and select **Add Field, as Child Node**. Repeat this to create Field1, Field2 and Field3.

11 Triple-click on **Field1**, and rename it **Order**.

12 Select the **Order** node. The properties for the Order node are displayed in the Properties pane. Change the **endDelim** property to "**|**" (pipe, without the quotes).

13 Triple-click on **Field2**, and rename it **LineItem**.

14 In the **LineItem** node Properties, **endDelim** field, enter "**|**" (pipe).

15 Triple-click on **Field3**, and rename it **Total**.

16 In the **Total** node Properties, **endDelim** field, enter "**|**" (pipe).

Figure 6 Event Type Definition Editor
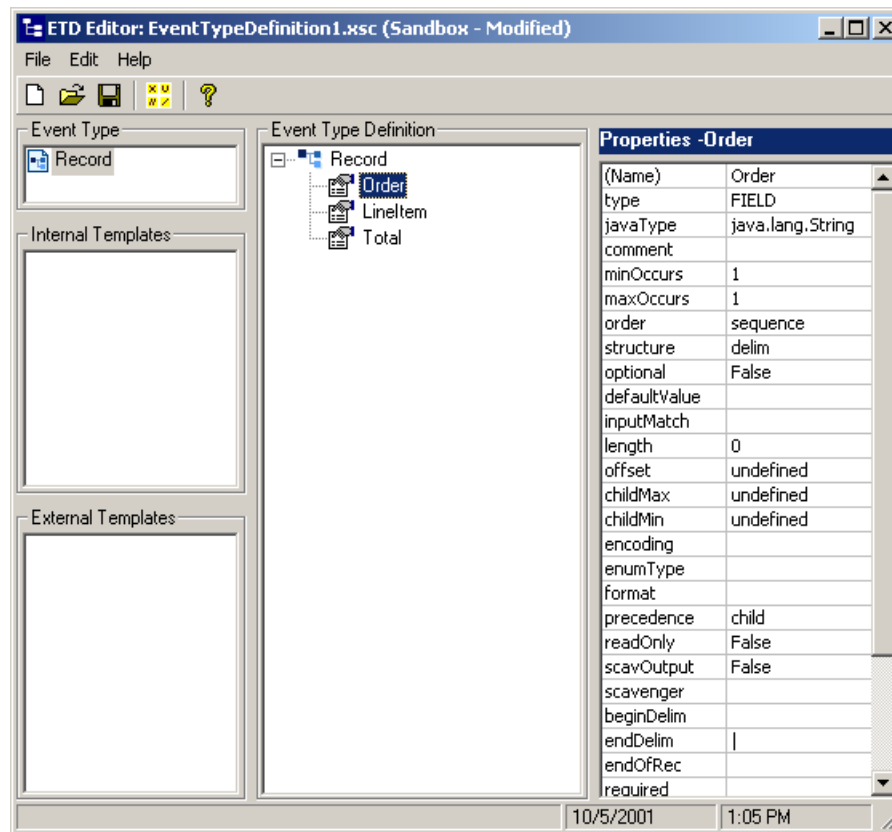
17 From the File menu, click **Compile and Save**. Save the .xsc file as **Record.xsc**.

18 From the File menu, click **Promote to Run Time** to promote the file to the run time environment.

19 Close the ETD Editor.

5.2.6 Step Six–Intelligent Queues

Step six in configuring the MQSeries e*Way is to create the IQs. IQs manage the exchange of information between components within the e*Gate system, providing non-volatile storage for data as it passes from one component to another. IQs use IQ Services to transport data. IQ Services provide the mechanism for moving Events between IQs, handling the low-level implementation of data exchange (such as system calls to initialize or reorganize a database).

To create and modify an Intelligent Queue for the MQSeries e*Way

1 Select the Navigator's **Components** tab.

2 Open the host on which you want to create the IQ.

3 Open a **Control Broker**.

4   Select an **IQ Manager**.

5   On the palette, click the **Create a New IQ** button.

6   Enter the name of the new IQ (in this case, **iq_1**), then click **OK.**

7   Double-click the new **IQ** to edit its properties.

8   On the **General** tab, specify the **Service** and the **Event Type Get Interval**.

   The **STC_Standard** IQ Service provides sufficient functionality for most applications. If specialized services are required, custom IQ Service DLLs may be created.

   The default **Event Type Get Interval** of 100 Milliseconds is satisfactory for the purposes of this initial implementation.

9   On the **Advanced** tab, make sure that **Simple publish/subscribe** is checked under the **IQ behavior** section.

10 Click OK to close the **IQ Properties** window.

11 For this schema, repeat steps 1 through 10 to create an additional IQ (**IQ_2**).

5.2.7 Step Seven: Collaboration Rules

Step seven in creating the MQSeries e*Way is to create the Collaboration Rules that will extract and process selected information from the source Event Type defined earlier, according to its associated Collaboration Service. The **Default Editor** can be set to either **Monk** or **Java**. From the **Enterprise Manager Task Bar,** select **Options** and click **Default Editor**. The default should be set to **Java**.

The sample schema requires the creation of two collaboration Rules files

   ▪ **"Creating Pass Through Collaboration Rules" on page 38**

   ▪ **"Creating Java Collaboration Rules" on page 39**

Creating Pass Through Collaboration Rules

1   Select the Navigator's **Components** tab in the e*Gate Enterprise Manager.

2   In the **Navigator**, select the **Collaboration Rules** folder.

3   On the palette, click the **Create New Collaboration Rules** button.

4   Enter the name of the new Collaboration Rule Component, then click **OK** (for this case, use **Pass**).

5   Double-click the new Collaboration Rules Component. The **Collaboration Rules Properties** window opens.

6   The **Service** field defaults to **Pass Through**.

Figure 7 Collaboration Properties
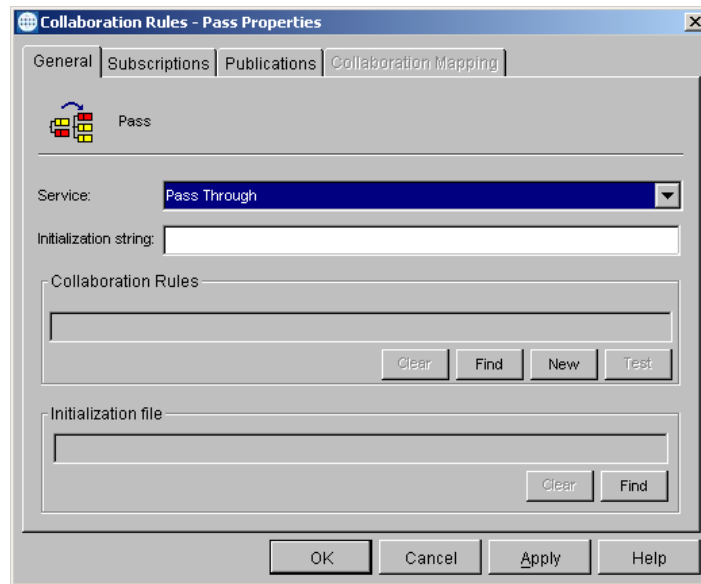
7   Go to the **Subscriptions** tab. Select **GenericInEvent** under **Available Input Event Types**, and click the right arrow to move it to **Selected Input Event Types**. The box under **Triggering Event** should be checked.

8   Go to the **Publications** tab. Select **GenericInEvent** under **Available Output Event Types**, and click the right arrow to move it to **Selected Output Event Types**. The Radio button under **Default** will be enabled.

9   Click **OK** to close the **Collaboration Rules, Pass Properties** window.

Creating Java Collaboration Rules

1   Select the Navigator's **Components** tab in the e*Gate Enterprise Manager.

2   In the **Navigator**, select the **Collaboration Rules** folder.

3   On the palette, click the **Create New Collaboration Rules** button.

4   Enter the name of the new Collaboration Rule, then click **OK** (for this case, use **JavaCollab**).

5   Double-click the new Collaboration Rules Component to edit its properties. The **Collaboration Rules Properties** window opens.

6   From the **Service** field drop-down box, select Java. The **Collaboration Mapping** tab is now enabled, and the **Subscriptions** and **Publications** tabs are disabled.

7   In the **Initialization string** field, enter any required initialization string for the Collaboration.

Figure 8 Collaboration Rules - JavaCollab Properties
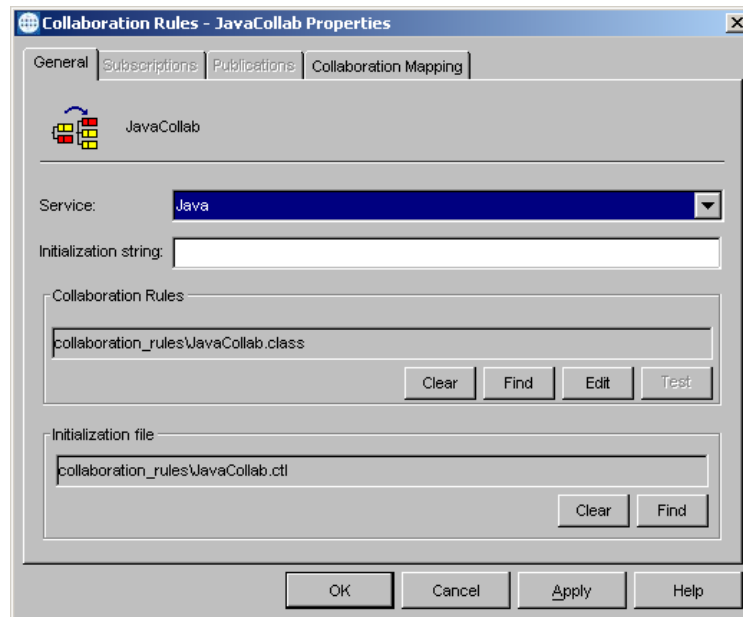
8 Select the **Collaboration Mapping** tab.

9 Using the **Add Instance** button, create instances to coincide with the Event Types.

   For this sample, do the following:

10 In the **Instance Name** column, enter **In** for the instance name.

11 Click **Find**, navigate to **etd\Record.xsc**, double-click to select. **Record.xsc** is added to the **ETD** column of the instance row.

12 In the **Mode** column, select **In** from the drop–down menu available.

13 In the **Trigger** column, click the box to enable trigger mechanism.

14 Repeat steps 9–13 using the following values:

   ◆ Instance Name — **Out**

   ◆ ETD — **Record.xsc**

   ◆ Mode — **Out**

Note: *At least one of the ETD instances used by the Collaboration must be checked as the trigger.*

   *For specific information on creating and configuring Collaboration Rules, see the e*Gate Integrator User's Guide.*

Figure 9   Collaboration Rules - Collaboration Mapping Properties
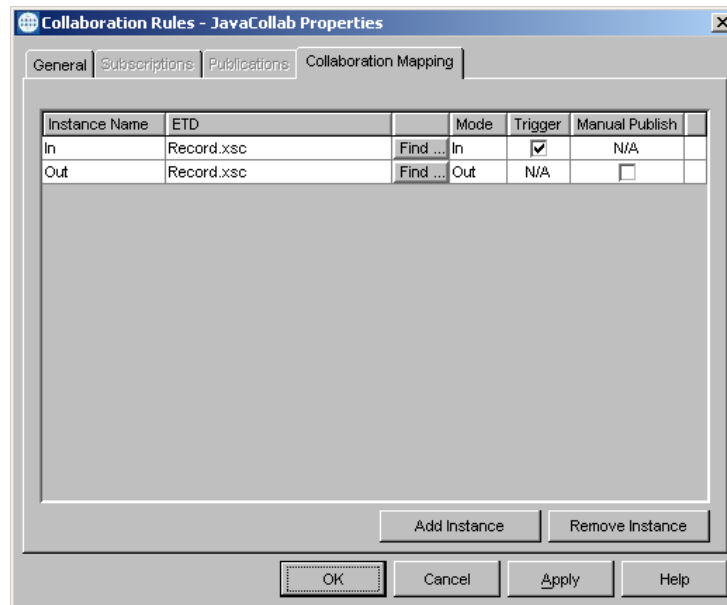
The "Read from MQSeries" is carried out by the following processes.

A   The **Event Type "Get" interval** polls for available messages at the prescribed interval.

B   The **receive()** method for an ETD associated with an MQSeries e*Way Connection is invoked, initiating a "read" on MQSeries.

15   Select the **General** tab, under the Collaboration Rule box, select **New**. The **Collaboration Rules Editor** opens.

16   Expand to full size for optimum viewing, expanding the Source and Destination Events as well.

5.2.8   Using the Collaboration Rules Editor

Part two of step seven is to define the business logic using the Collaboration Rules Editor. The Java Collaboration Rules Editor is the GUI used to create and modify Java Collaboration Rules. A Java Collaboration Rule is created by designating one or more source Events and one or more destination Events and then setting up rules governing the relationship between fields in the Event instances.

Creating the Collaboration Rules Class

1   Highlight **retBoolean** in the **Business Rules** pane.

All of the user–defined business rules are added as part of this method.

2   Select **Order** from the **Source Events** pane. Drag–and–drop onto **Order** in the **Destination Events** pane. A connecting line appears between the properties objects.

3   In the **Business Rules** pane, a rule expression appears, with the properties of that rule displayed in the **Rule Properties** pane.

4 Select **LineItem** from the **Source Events** pane. Drag–and–drop onto **LineItem** in the **Destination Events** pane.

5 Select **Total** from the **Source Events** pane. Drag–and–drop onto **Total** in the **Destination Events** pane.

Figure 10   Collaboration Rules — Collaboration Rules Editor
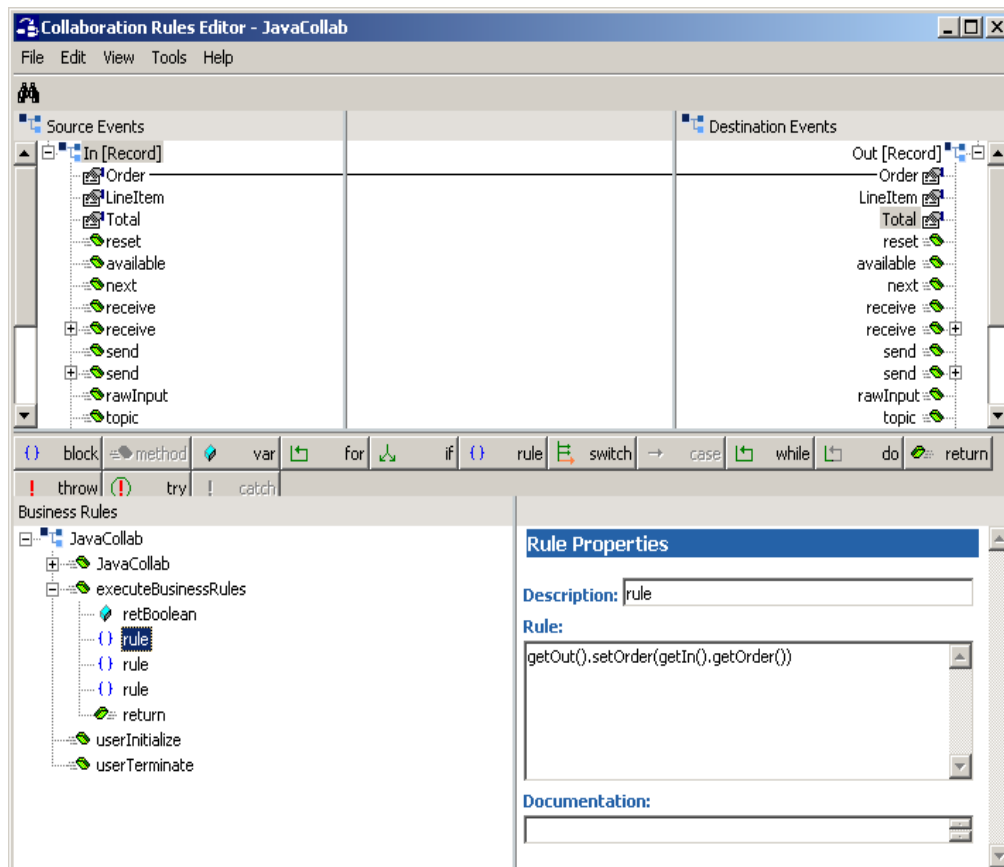
6 When all the business logic has been defined, the code can be compiled by selecting **Compile** from the **File** menu. The **Save** menu opens, provide a name for the **.xpr** file. For the sample, use **MQSSample.xpr**.

If the code compiles successfully, the message **Compile Completed** appears. If the outcome is unsuccessful, a Java Compiler error message appears.

Once the compilation is complete, save the file and exit.

7 Under the **Collaboration Rules**, the path for the .class file created appears. (For the sample, the path **"collaboration_rules\JavaCollab.class "**appears.)

8 Under **Initialization** file, the path for the **.ctl** file created appears. (For the sample the path **"collaboration_rules\JavaCollab.ctl"** appears.)

9 Click **OK** to exit the **Properties** Box.

Note: *For detailed information on creating Collaboration Rules using the Java Collaboration Rules Editor see the e*Gate Integrator User's Guide.*

5.2.9 Step Eight: Collaborations

Step eight in creating the MQSeries e*Way is to create the Collaborations. Collaborations are the components that receive and process Event Types, then forward the output to other e*Gate components or an external component. Collaborations consist of the Subscriber, which "listens" for Events of a known type (sometimes from a given source), and the Publisher, which distributes the transformed Event to a specified recipient.

Create the Inbound_eWay Collaboration

1 In the e*Gate Enterprise Manager, select the Navigator's **Components** tab.

2 Open the host on which you want to create the Collaboration.

3 Select a **Control Broker.**

4 Select the **ew_In** e*Way to assign the Collaboration.

5 On the palette, click the **Create a New Collaboration** button.

6 Enter the name of the new Collaboration, then click **OK.** (For the sample, "**In_cr**".)

7 Double-click the new **Collaboration** to edit its properties.

8 From the **Collaboration Rules** list, select the **Collaboration Rules** file that you created previously (for this sample, "**Pass**").

9 In the **Subscriptions** area, click **Add** to define the input Event Types to which this Collaboration will subscribe.

   A From the **Event Type** list, select the **Event Type** that you previously defined **GenericInEvent**.

   B Select the **Source** from the **Source** list. In this case, it should be **<External>**.

10 In the **Publications** area, click **Add** to define the output **Event Types** that this Collaboration will publish.

   A From the **Event Types** list, select the **Event Type** that you previously defined **GenericInEvent**.

   B Select the publication **Destination** from the **Destination** list. In this case, it should be **iq_1**.

   C The Priority column will default to **5**.

Figure 11 Collaboration - Inbound e*Way Properties
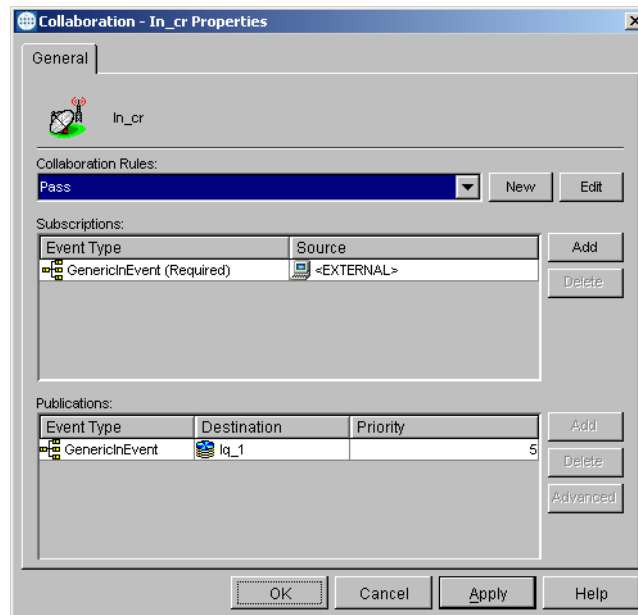
11 Click **OK** to close the **Collaboration Properties** window.

Create the MQ_stceway Multi Mode e*Way Collaborations

Two Collaboration will be created for the Multi-Mode e*Way **MQ_cr_out**, and **MQ_cr_in**.

1 To create the **MQ_cr_out Collaboration**, Select the **MQ_stceway** e*Way to assign another Collaboration.

2 On the palette, click the **Create a New Collaboration** button.

3 Enter the name of the new Collaboration, then click **OK.** (For the sample, "**MQ_cr_out**".)

4 Double -click the new Collaboration to edit its properties.

5 From the **Collaboration Rules** list, select the **Collaboration Rules** file that you created previously. For the sample use **JavaCollab.**

6 In the **Subscriptions** field, click **Add** to define the input Event Types to which this Collaboration will subscribe.

   A From the **Instance Name** list, select the **Instance Name** that you previously defined **In**.

   B From the **Event Type** list, select the **Event Type** previously defined **GenericInEvent**.

   C Select the **Source** from the **Source** list. In this case, it should be **In_cr**.

7 In the **Publications** area, click **Add** to define the output **Event Types** that this Collaboration will publish.

   A From the **Instance Name** list, select the **Instance Name** previously defined **Out**.

B From the **Event Types** list, select the **Event Type** that you previously defined **Ev_1**.

Important: *The Event Type name must be the same as the IBM MQSeries queue name.*

C Select the **Destination** from the **Destination** list. In this case, it should be **MQ_conn1**.

D The Priority column will default to **5**.

Figure 12 Collaboration Properties - MQ_cr_out
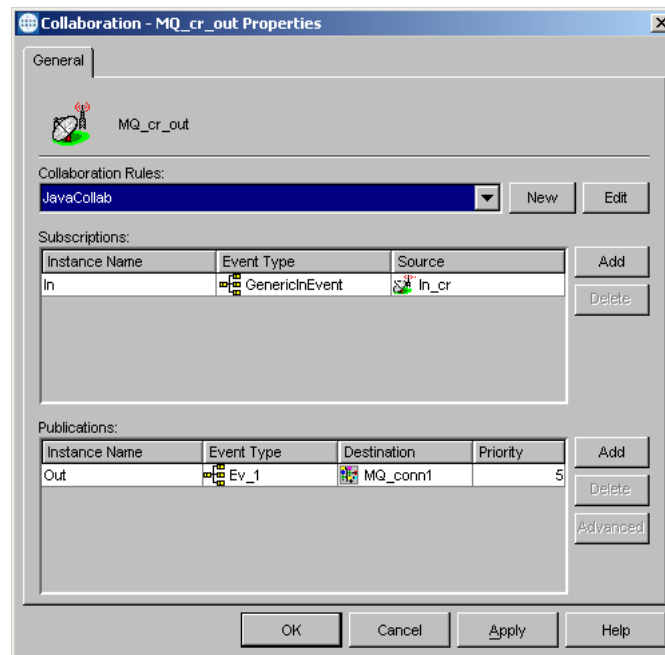
8 Click **OK** to close the Properties window.

9 To create the **MQ_cr_in collaboration**, select the Navigator's **Components** tabIn the e*Gate Enterprise Manager.

10 Open the host on which you want to create the Collaboration.

11 Select a **Control Broker.**

12 Select the **MQ_stceway** e*Way to assign the Collaboration.

13 On the palette, click the **Create a New Collaboration** button.

14 Enter the name of the new Collaboration, then click **OK.** (For the sample, "**MQ_cr_in**".)

15 Double-click the new **Collaboration** to edit its properties.

16 From the **Collaboration Rules** list, select the **Collaboration Rules** file that you created previously. For the sample use **JavaCollab.**

17 In the **Subscriptions** field, click **Add** to define the input Event Types to which this Collaboration will subscribe.

    A From the **Instance Name** list, select the Instance Name that you previously defined (**In**).

    B From the **Event Type** list, select the **Event Type** previously defined **Ev_1**.

    C Select the **Source** from the **Source** list. In this case, it should be **MQ_conn1**.

18 In the **Publications** area, click **Add** to define the output **Event Types** that this Collaboration will publish.

    A From the **Instance Name** list, select the **Instance Name** previously defined **Out**.

    B From the **Event Types** list, select the **Event Type** that you previously defined (**GenericInEvent**).

    C Select the publication destination from the **Destination** list. In this case, it should be **iq2**.

19 Click **OK** to close the Collaboration window.

Figure 13 Collaboration Properties - MQ_cr_in
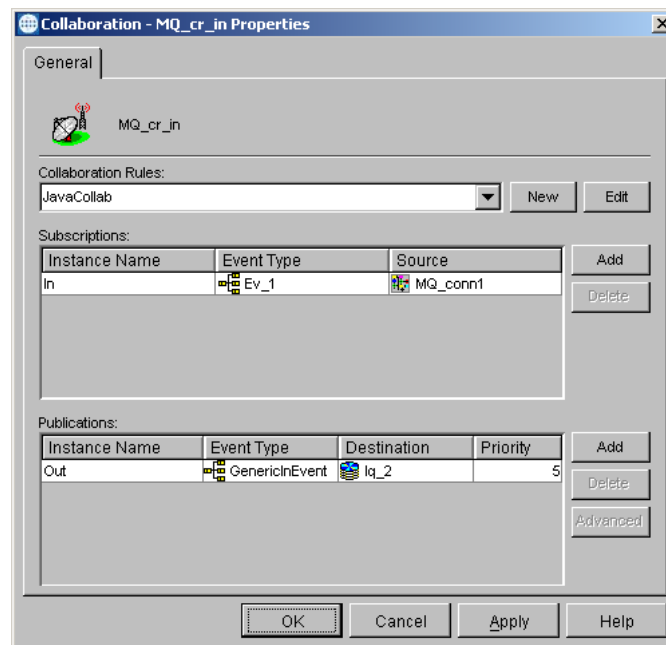
20 Click **OK** to exit.

Create the Outbound_eWay Collaboration

1 In the e*Gate Enterprise Manager, select the Navigator's **Components** tab.

2 Open the host on which you want to create the Collaboration.

3 Select a **Control Broker.**

4 Select the **ew_Out** e*Way to assign the Collaboration.

5 On the palette, click the **Create a New Collaboration** button.

6 Enter the name of the new Collaboration, then click **OK.** (For the sample, "**Out_cr**".)

7 Double-click the new **Collaboration** to edit its properties.

8   From the **Collaboration Rules** list, select the **Collaboration Rules** file that you previously defined **Pass**.

9   In the **Subscriptions** area, click **Add** to define the input Event Types to which this Collaboration will subscribe.

   A   From the **Event Type** list, select the **Event Type** that you previously defined **GenericInEvent**.

   B   Select the **Source** from the **Source** list. In this case, it should be **MQ_cr_in**.

10 In the **Publications** area, click **Add** to define the output **Event Types** that this Collaboration will publish.

   A   From the **Event Types** list, select the **Event Type** that you previously defined **GenericInEvent**.

   B   Select the publication destination from the **Destination** list. In this case, it should be **<External>**.

   C   Click **OK** to close the **Collaboration Properties** window.

5.2.10 Step Nine: Setting CLASSPATH Variable

The final step in creating and configuring the MQSeries e*Way is to set the IBM MQSeries Java .jar files in the environment CLASSPATH variable. This includes the following .jar files.

   \MQSeries\Java\lib

   \MQSeries\Java\lib\providerutil.jar

   \MQSeries\Java\lib\ldap.jar

   \MQSeries\Java\lib\jndi.jar

   \MQSeries\Java\lib\com.ibm.fscontext.jar

   \MQSeries\Java\lib\com.ibm.mqjms.jar

   \MQSeries\Java\lib\com.ibm.mqbind.jar

   \MQSeries\Java\lib\com.ibm.mq.jar

   \MQSeries\Java\lib\com.ibm.mq.iiop.jar (com.ibm.mq.iiop.jar only applies for Windows, not UNIX.)

Also set the \MQSeries\Java\lib in your PATH.

For Unix, include /MQSeries/Java/lib in the library path as follows:

   ▪ Solaris: LD_LIBRARY_PATH

   ▪ HP-UX: SHLIB_PATH

   ▪ AIX: LIBPATH

If the CLASSPATH and PATH already exist, add the .jar files to the existing PATH and CLASSPATH.

Setting CLASSPATH variable on Windows

To set the .jar files from java classes in classpath do the following:

1 Right-click **My Computer** and select **Properties**. The System Properties window opens.

2 Select the **Advanced** tab and Click on **Environment Variables**. The Environment Variables window opens.

3 Under System Variables click the **New** button.

4 In the New System Variable window type **ClassPath** in the **Variable Name** field. In the Variable Value field type the absolute path for the first .jar file (See figure 12), and click **OK**.

Figure 14 Set Environment Variables

5 Repeat steps 3 and 4 for each of the MQSeries .jar files.

6 Under System Variables click the **New** button.

7 In the New System Variable window type **Path** in the **Variable Name** field. In the Variable Value field type the absolute path for \MQSeries\Java\lib and click **OK**.

8 Click **OK** to close the Environment Variables window and the System Properties window.

5.2.11 Execute the Schema

To execute the MQSeries sample schema, do the following

1 Go to the command line prompt, and enter the following:

```
stccb -rh hostname -rs schemaname -un username -up user password
-ln hostname_cb
```

Substitute *hostname*, *schemaname*, *username* and *user password* as appropriate.

2 Exit from the command line prompt, and start the e*Gate Monitor GUI.

3 When prompted, specify the *hostname* which contains the Control Broker you started in Step 1 above.

4 Select the MQSeries sample schema.

5 After you verify that the Control Broker is connected (the message in the Control tab of the console will indicate command *succeeded* and status as *up*), highlight the IQ Manager, *hostname*_igmgr, then right-click and select **Start**.

6 Highlight each of the e*Ways, right-click the mouse, and select **Start**.

5.2.12 Error Messages

If there is an error, such as a failed connection, an exception is thrown by the module and logged to error log file at egate/client/logs. The error log will appear similar to the following:

```
11:59:34.091 EWY I 11 (initialize.cxx:1035): Exception thrown: Failed to access queue:
MQRC_UNKNOWN_OBJECT_NAMEc
om.ibm.mq.MQException: MQJE001: Completion Code 2, Reason 2085
    at com.ibm.mq.MQQueueManager.accessQueue(MQQueueManager.java:1151)
    at com.ibm.mq.MQQueueManager.accessQueue(MQQueueManager.java:1196)
    at com.stc.eways.MQSeriesETD.MQSeriesConnector.accessQueue(MQSeriesConnector.java:395)
    at com.stc.eways.MQSeriesETD.MQSeriesETD.accessQueue(MQSeriesETD.java:291)
    at MQ_EMECollab.executeBusinessRules(MQ_EMECollab.java:106)
    at com.stc.jcsre.JCollaboration.translate(JCollaboration.java:97)
    at com.stc.common.collabService.JCCollabControllerImpl.
translate(JCCollabControllerImpl.java:1096
```

The reason code parameter or MQRC, in this case Reason 2085, appears in the first few lines of the error log. This reason code can be used in conjunction with IBMs online document, **MQSeries Messages**, Chapter 9 at:

http://www-903.ibm.com/board/attach_files/mqseries/k1005706457257_messages.pdf

The chapter lists reason codes, exceptions, the associated errors and the corrective actions to take. For the above example, the MQRC appears as follows:

2085	X'0825'	MQRC_UNKNOWN_OBJECT_NAME
		An MQOPEN or MQPUT1 call was issued, but the object identified by the ObjectName and ObjectQMgrName fields in the object descriptor MQOD cannot be found. One of the following applies:
		▪ The ObjectQMgrName field is one of the following:
		◆ Blank
		◆ The name of the local queue manager
		◆ The name of a local definition of a remote queue (a queue-manager alias) in which the RemoteQMgrName attribute is the name of the local queue manager but no object with the specified ObjectName and ObjectType exists on the local queue manager.
		▪ The object being opened is a cluster queue that is hosted on a remote queue manager, but the local queue manager does not have a defined route to the remote queue manager.
		▪ The object being opened is a queue definition that has QSGDISP(GROUP). Such definitions cannot be used with the MQOPEN and MQPUT1 calls.
		Corrective action: Specify a valid object name. Ensure that the name is padded to the right with blanks if necessary. If this is correct, check the queue definitions.

ETD Overview

This chapter gives an overall view of the MQSeriesETD hierarchy structure, including available methods and properties, and their application. For a more detailed description of each method see **MQSeries Java Methods (ETD)** on page 81

6.1 The MQSeriesETD

The following is the general outline of the ETD and the methods and properties exposed on each node. Any methods noted with *, are methods or properties above and beyond the exposed base MQSeries java API. Ellipses for the parameters indicate one or more arguments for the method. The purpose of each is explained below.

```
+ QueueManager
      void connectToQueueManager(...)*
      void selectQueueManager(String name)*
      boolean isQueueMgrConnected()
      int getCharacterSet()
      int getMaximumPriority()
      void commit()
      void backout()
      queueAccessOptions*
      accessQueue(name)*
      + Queue
            + GMO *
            + PMO *
            selectQueue(name)*
            void get()
            void getWithOptions()*
            void put()
            void putWithOptions()*
            int getCurrentDepth()
            int getMaximumDepth()
            int getMaximumMessageLength()
            void newMessage()*
            + Message
                  + MsgHeader*
                        all properties
                  + MsgBody*
```

byte[] Data*
readData()*
writeData()*
all the methods on MQMessage

6.1.1 The QueueManager Node

QueueManager is the root node and represents the interface to the MQQueueManager object in the MQSeries API. The name of the node, **QueueManager** as opposed to **MQSeriesETD** is used as a descriptive way of conveying the exposed hierarchy. Internally, it is actually implemented in the ETD implementation class com.stc.eways.MQSeriesETD.

The ETD implementation class will hold a collection of queue manager objects to allow the user to connect to more than one queue manager and, once connections have been established, to switch between them. An exception to this is when the e*Way is configured as XA compliant. In this case, there can be exactly one queue manager which is the one specified in the configuration.

Current Queue Manager

Connecting to a queue manager will automatically select it as the **current queue manager**. If you have connected to more than one queue manager, you can switch between them using the **selectQueueManager** method. Each queue manager is accessible via its name. When the collaboration is initialized, it automatically connects to the queue manager specified in the configuration (which, again, selects it as the current queue manager). Thus, if you do not connect to another queue manager in the collaboration, you need not ever call the selectQueueManager method.

The queueAccessOptions Node

The **queueAccessOptions** node and the **accessQueue** method are used to access a queue on the current queue manager. First the desired queue access options are entered (such as open for input or open for output, and so forth), and then the **accessQueue** method is called. This method will access the named queue on the current queue manager and select that queue as the current queue.

The remaining methods exposed on the queue manager route directly to the similarly named method on the queue manager in the underlying MQSeries API.

Methods Under the QueueManager Node

Name	Description
connectToQueueManager on page 97	Create a connection to the another queue manager using the specified parameters. A connection to the queue manager specified in the configuration is automatically done. You need only call this method when connecting to another queue manager in the collaboration.
selectQueueManager on page 97	Select from one of the connected queue managers.

Name	Description
isQueueMgrConnected on page 98	Determine if the current queue manager is still connected.
getCharacterSet on page 98	Returns CCSID of the queue managers codeset for the currently selected queue manager.
getMaximumPriority on page 99	Returns maximum message priority that can be handled by the queue manager.
commit on page 99	Commit the operations on the currently selected queue manager. Should only be called in Non-XA mode.
backout on page 99	Roll back the operations on the currently selected queue manager. Should only be called in Non-XA mode.
queueAccessOptionsClearAll on page 100	Clear all flags.
accessQueue on page 100	Access a queue on the current queue manager.

Note: *In MQSeries java e*Way, it is important that the queue access options are set before attempting to access the queue. The MQ java code will throw an exception if the options are not set first. You can liken this to pressing the gas pedal in a car before putting it in a gear; it'll go no where.*

6.1.2 The Queue Node

The Queue node corresponds to operations that are performed on the MQQueue object in the MQSeries API. In the ETD, it is shown as a child of the QueueManager node. This is to enforce the concept that you access a queue from a queue manager. As in the QueueManager node, the node name of Queue is a notational convenience for the user. Internally, it is implemented in the MQSeriesETD class.

Current Queue

The ETD uses the concept of a **current queue**. This is not the same as the Current Queue Manager concept noted earlier. The ETD will support accessing one or more queues from the current queue manager via the **accessQueue** function on the **QueueManager** node. Calling this will also select that queue as the current queue. The user can also select different queues (which have already been accessed from the queue manager) by using the **selectQueue** function. Selecting a queue sets it as the current queue. Early in the collaboration code, it is typical that the user would call **accessQueue** to access the queue to be used in the collaboration. The ETD does not automatically access a queue at initialization time as it does for the queue manager. It is important to remember that all the methods take effect on whichever queue or queue manager is current. For example, calling the put method to put a message on the queue takes effect on the current queue.

Get and Put Methods

There are two versions of the **get** and **put** methods. Each operates on the message object exposed in the ETD. Each routes directly to the corresponding underlying method on the queue in the MQSeries API. One of the **get** and **put** methods take no arguments; they use the default options in MQSeries, whatever these may be. The other uses the **GetMessageOptions** (the **GMO**) or the **PutMessageOptions** (the **PMO**). These options allow the user to set whatever options they wish (for example, the SET_ALL_CONTEXT flag).

These classes contain options that control the action of the getWithOptions and putWithOptions methods. The options are mostly bitfields in the MQSeries API. For example, the MQC.MQGMO_WAIT and the MQC.MQGMO_SYNCPOINT are two bitfields that can be set for the "options" member variable in the MQGetMessageOptions class. To make it easier for the user, these bitfields have been expanded into callable methods that take a boolean parameter to set or clear the particular option. For example, if you want to set a wait timeout value in the GetMessageOptions, you would call setMQGMO_WAIT(true) and then call the setWaitValue method. If you wanted this message to be a syncpoint, you would call setMQGMO_SYNCPOINT(true). Correspondingly, if you wanted to clear the syncpoint flag, you would call setMQGMO_SYNCPOINT(false) then call setMQGMO_NO_SYNCPOINT(true).

The optionsClearAll (and matchOptionsClearAll and so forth) allow you to clear all previously set options with one method.

Whatever values you set in the GMO and PMO nodes remain in effect for the duration of the collaboration. That is, if you are putting more than one message, and they both take the same PutMessageOptions, you only need to set the options once.

Note, too, that some of these flags may be required in certain contexts. An example is the SYNCPOINT flag when using transactions. In this case, calling commit in the ETD (or when the XA transaction is committed by e*Gate) the SYNCPOINT flag MUST be set.

Note: *Some of the members of these classes may be shown as output fields in the API documentation. In such cases, you should not attempt to set a value on them as they are populated by the underlying API method.*

Important: *Some of the attributes of the message header, such as userId, are affected by the* ***SET_ALL_CONTEXT*** *flag in the* ***PutMessageOptions****. Without SET_ALL_CONTEXT set, MQSeries will overwrite whatever value you might put in. If the flag is set, however, MQSeries will pass on the value you entered untouched.*

The newMessage Method

The **newMessage** method allows the user to destroy and recreate a new Message object. This is required when you want to call **get** multiple times in the collaboration (such as in a loop). If **get** is called again, passing a "dirty" message, the API will throw an exception indicating no message is available.

The remaining methods on the queue route directly to the similarly named method on the underlying queue object in the MQSeries API. As a application note, be careful with the "interrogative" type methods on the queue such as **getCurrentDepth**. In order to call this, the queue must be accessed with **MQOO_INQUIRE** set.

Methods Under the Queue Node

Name	Description
optionsClearAll on page 104	Clear all option flags.
setWaitValue on page 105	Specify a specific number of milliseconds to wait.
setUnlimitedWait on page 105	Set the wait time to MQWI_UNLIMITED.
matchOptionsClearAll on page 106	Clear all match options flags set so far and set match options to MQMO_NONE.
get on page 101	Get a message off the queue using the default options.
getWithOptions on page 101	Get a message off the queue using the GetMessageOptions (GMO).
put on page 102	Put a message on the queue using the default options.
putWithOptions on page 102	Put a message on the queue using the PutMesageOptions (PMO).
getCurrentDepth on page 102	Get the number of messages currently in the queue.
getMaximumDepth on page 103	Get the maximum number of messages that can exist on the current queue.
getMaximumMessageLength on page 103	Get the maximum length of data that can exist in any one message on the current queue.
newMessage on page 104	Destroy then recreate the Message object. After doing a get, this must be called before doing another get.

6.1.3 The Message Node

The Message node corresponds to methods that are called on the message object in the MQSeries API. It is shown as a child of the Queue node in the ETD to enforce the concept that it is rather subservient to the queue. That is, you get and put messages from and to a queue.

The MsgHeader Child Node

The MsgHeader child node of the Message wraps the concept of the attributes of the **MQMessage**. There is no concept of a message header in the MQSeries API per se. Rather, this is a notational convenience in the ETD. By using the nodes of the **MsgHeader**, you can gain access to the attributes of the message (that is **userId**, **msgId**, and so forth).

The MsgBody Child Node

The **MsgBody** child node of the Message wraps the concept of exposing the message data as a byte array. There is no corresponding "body" concept in the MQSeries API. That is, the only way to gain access to the data in the message is by calling one of the read methods. MsgBody is a notational convenience for the user to allow them to access the entire data of the message as a blob and have that blob stored in a node in the ETD. To access the message data after doing a get, the **readData** method is called. This routes down to the **readFully** method on the message. The **Data** node is then populated with the entire contents of the message. The data is now available from the **Data** node so you can drag it to somewhere else in the collaboration. To put a blob on the queue, data is "dragged and dropped" to the Data node and the **writeData** method on the MsgBody is called. The **writeData** method on the **MsgData** node routes down to the write method on the message.

Calling Read Methods

There are some important application caveats when dealing with the MQSeries API when it comes to calling the read methods. As you call a read method (for example. **readUTF**, **readInt**, and so forth) you are "consuming" the message data. Should you continue to call them, you will eventually exhaust the available data of the message and you will ultimately get an **EOFException**. This indicates you have reached the end of the data. This exception is caught in the underlying implementation code of the e*Way and is rethrown as a **CollabConnException** but the **EOFException** is still available. The reason for the exception is logged. Therefore, if you want to re-read a portion of the data you need to call seek to put the current data offset back. For example, if readFully is called, all data is consumed. If you want to call readFully again you need to do a **seek(0)**.

The MQMessage Class

The remaining methods on the Message node all route down to the similarly named methods on the MQMessage class. A brief description of each method is available in the Properties field of the ETD Editor when the method is selected.

Methods Under the Message Node

Name	Description
getTotalMessageLength on page 107	If MQQueue.get() fails with a message-truncated error code, report the total number of bytes in the stored message on the queue.
getMessageLength on page 108	Report the total number of bytes in the stored message on the queue.
getDataLength on page 108	Report the number of bytes of data remaining to be read in the message.
seek on page 109	Relocate the cursor to the absolute position in the message buffer given by pos.

Name	Description
setDataOffset on page 109	Relocate the cursor to the absolute position in the message buffer. setDataOffset ( ) is equivalent to seek(), allowing for cross-language compatibility with the other MQSeries APIs.
getDataOffset on page 110	Return the current position of the cursor within the message, that is the point at which read and write operations take effect.
clearMessage on page 110	Discard data in the message buffer and reset the data offset to zero.
getVersion on page 110	Return the version of the current structure.
resizeBuffer on page 111	Clue the MQMessage object as to the size of buffer that may be necessary for subsequent get operations. When a message contains message data, and the new size is less than the current size, the message data is truncated.
readBoolean on page 111	Read a (signed) byte from the present position in the message buffer.
readChar on page 112	Read a Unicode character from the present position in the message buffer.
readDouble on page 112	Read a double from the present position in the message buffer.
readFloat on page 112	Read a float from the present position in the message buffer.
readFully on page 113	Fill the byte array b with data from the message buffer. Fill len elements of the byte array b with data from the message buffer, starting at offset off.
readInt on page 114	Read an integer from the present position in the message buffer.
readInt4 on page 114	Equivalent to readInt(), provided for cross-language MQSeries API compatibility.
readLine on page 114	Converts from the codeset defined in the characterSet member variable to Unicode, then reads in a line that has been terminated by \n, \r, \r\n, or EOF.
readLong on page 115	Read a long from the present position in the message buffer.
readInt8 on page 115	Equivalent to readlong(), provided for cross-language MQSeries API compatibility.
readObject on page 116	Read an object, its class, class signature, and the value of the non-transient and non-static fields of the class.
readShort on page 116	Read a short from the present position in the message buffer.

Name	Description
readInt2 on page 117	Equivalent to readshort(), provided for cross-language MQSeries API compatibility.
readUTF on page 117	Read a UTF string, prefixed by a 2-byte length field, from the present position in the message buffer.
readUnsignedByte on page 117	Read an unsigned byte from the present position in the message buffer.
readUnsignedShort on page 118	Read an unsigned short from the present position in the message buffer.
readUInt2 on page 118	Equivalent to readUnsignedShort(), provided for cross-language MQSeries API compatibility.
readString on page 119	Read a string in the codeset defined by the characterSet member variable. Convert the string into Unicode.
readDecimal2 on page 119	Read a 2-byte packed decimal number.
readDecimal4 on page 120	Read a 4-byte packed decimal number.
readDecimal8 on page 120	Read a 8-byte packed decimal number.
setVersion on page 120	Sets the version of the structure to be used.
skipBytes on page 121	Advance n bytes in the message buffer. Block until all the bytes are skipped, the end of message buffer is detected, or an exception is thrown.
write on page 121	Write a byte, an array of bytes, or a series of bytes into the message buffer at the present position. len bytes will be written, taken from offset off in the array b.
writeBoolean on page 122	Write a boolean into the message buffer at the present position.
writeByte on page 123	Write a byte into the message buffer at the present position.
writeBytes on page 123	Writes the string to the message buffer as a sequence of bytes. Each character is written out in sequence by discarding its high eight bits.
writeChar on page 124	Write a Unicode character into the message buffer at the present position.
writeChars on page 124	Write a string as a sequence of Unicode characters into the message buffer at the current position.
writeDouble on page 124	Write a double into the message buffer at the present position.
writeFloat on page 125	Write a float into the message buffer at the present position.
writeInt on page 126	Write an integer into the message buffer at the present position.

Name	Description
writeLong on page 126	Write a long into the message buffer at the present position.
writeObject on page 127	Write the specified object, object class, class signature, and the values of the non-transient and non-static fields of the class and all its supertypes.
writeShort on page 127	Write a short into the message buffer at the present position.
writeDecimal2 on page 128	Write a 2-byte packed decimal format number into the message buffer at the present position.
writeDecimal4 on page 128	Write a 4-byte packed decimal format number into the message buffer at the present position.
writeDecimal8 on page 129	Write an 8-byte packed decimal format number into the message buffer at the present position.
writeUTF on page 129	Write a UTF string, prefixed by a 2-byte length field, into the message buffer at the present position.
writeString on page 130	Write a string into the message buffer at the present position, converting it to the codeset identified by the characterSet member variable.

6.1.4 Exception Handling

A general note on exception handling when using the ETD in a collaboration: Internally, the e*Way catches *all* of the **MQExceptions** thrown from the underlying MQSeries API. It also catches all other possible exception types that could be thrown from the MQSeries API methods (for instance, the **EOFException** on the message readXXX methods). The reason for this is to prevent users from having to do multiple catch clauses in their collaboration. It is only necessary to catch a total of two possible types of exceptions from the e*Way; the **CollabConnException** or the **EBobConnectionException**. The only time the **EBobConnectionException** is thrown by the eWay is in the initialization and shutdown phase of the eWay.

While the eWay catches all **MQExceptions** internally, the original exception is still available from the **CollabConnException**. Further, the MQExceptions "**reasonCode**" is logged in a human readable format. That is, instead of a cryptic numeric ID, it would log MQRC_NO_MESSAGE_AVAILABLE.

e*Way Connection Configuration (ETD)

This chapter describes how to configure the Java-enabled MQSeries *Way Connection using the MQSeriesETD connection type.
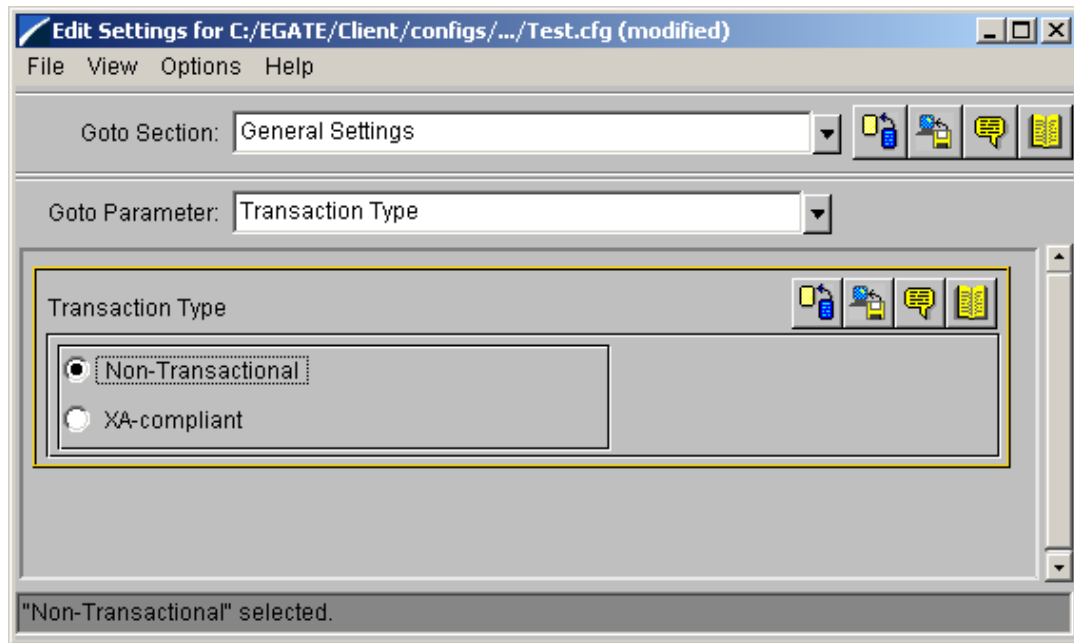
7.1 Configuring e*Way Connections

e*Way Connections are set using the Enterprise Manager.

To create and configure e*Way Connections

1 In the Enterprise Manager's **Component** editor, select the **e*Way Connections** folder.

2 On the palette, click on the **Create a New e*Way Connection** button.

3 The **New e*Way Connection Component** dialog box opens, enter a name for the **e*Way Connection**.

4 Double-click on the new **e*Way Connection**. The **e*Way Connection Properties** dialog box opens.

5 From the **e*Way Connection Type** drop-down box, select **MQSeriesETD**.

6 Enter the **Event Type "get" interval** in the dialog box provided. 100 milliseconds is the configured default. The "get interval is the intervening period at which, when subscribed to, the e*Way connection is polled.

7 From the **e*Way Connection Configuration File**, click **New** to create a new Configuration File for this e*Way Connection. (To use an existing file, click **Find**.)

8 The **Edit Settings** dialog box opens. Enter the correct parameters for your e*Way Connection as defined in the following pages.

9 When all parameters have been entered, select **File**, **Save** to save the settings, and **File**, **Promote to Run Time** to move the file to the runtime environment.

Figure 15 Edit Settings

The MQSeries e*Way Connection configuration parameters are organized into the following sections.

- **General Settings**

- **MQSeries**

- **Connector**

- **Default GetMessageOptions**

7.1.1 General Settings

This section contains the following parameters:

- **Transaction Type**

Transaction Type

Description

String-set. Specifies the Transaction Type. There are two transaction types.

- **Non-Transactional**. Provides the highest level of performance, with the minimum level of message protection. No rollback is available during the send and receive period, causing the possible loss of data in the case of a system error. When the send( ) method is called the transaction is immediate. Non-Transactional (single-phase transaction) relys on the user to call the commit and backout methods.

- **XA-compliant**. (two-phase transactional behavior) Highest level of transaction protection, providing rollback for internal and XA compliant transactions. The transaction is also extended to other XA supported data exchange applications,

such as Oracle, DB2, and MQSeries. When the send( ) method is called the transaction takes place at the end of the collaboration.

XA can only be used in Bindings mode. This means that a Host Name is not entered in the configuration parameters. A Host Name in the configuration parameters implies that the API is setup in Client mode. XA cannot be used in Client mode. In addition, when using XA:

- There must only be one Queue Manager.

- The method connectToQueueManager cannot be called.

- The methods commit and backout cannot be called. In Bindings mode e*Gate is the transaction coordinator and is in control of these methods.

Note: *Consult the XA Processing chapter of the e*Gate User's Guide for information on XA use and restrictions.*

Required Values

Select Non-Transactional or XA-compliant. Non-Transactional is the configured default.

7.1.2 MQSeries

This section contains the following top level parameters:

- **Queue Manager Name**
- **Host Name**
- **Port Number**
- **Channel**

Queue Manager Name

Description

String-set. Specifies the name of the IBM MQSeries Queue Manager to which the e*Way is to connect.

Required Values

Enter the name of the valid IBM MQSeries Queue Manager.

Host Name

Description

String-set. Specifies the name of the host on which the queue manager resides.

Note: *If the Host Name field is left blank, the e*Way will attempt to connect to MQSeries in Bindings mode. The Queue Manager must be on the local machine.*

Required Values

A valid host name.

Port Number

Description

Integer-set. Specifies the port number to which the queue manager is set to listen.

Required Values

An integer in the range of 1000 to 65536. The configured default is 1414.

Channel

Description

String-set. Specifies the name of the channel being used.

Required Values

The valid name of the channel.

7.1.3 Connector

This section contains the following top level parameters:

Note: *These parameters are used internally by the e*Way and are for future expansion potentialities. The default values should always be used.*

- **type**
- **class**
- **Property.Tag**

type

Description

String-set. Specifies the connector type for MQSeriesETD. The default value should always be used.

Required Values

A valid connector type. The default value should always be used. The configured default is MQSeriesETD

class

Description

String-set. Specifies the connector class for MQSeriesETD. The default value should always be used.

Required Values

The configured default is "com.stc.eways.MQSeriesETD.MQSeriesConnector".

Property.Tag

Description

Specifies the data source identity. This parameter is required by the current EBobConnectorFactory.

Required Values

A valid data source package name.

7.1.4 Default GetMessageOptions

This section contains the following top level parameters:

- **Wait Timeout**
- **Wait Interval**

Wait Timeout

Description

String-set. Specifies the time to wait for a message to arrive on the queue when calling getWithOptions.

- **Unlimited**. Wait forever.
- **No-Wait**. Return immediately if no message is available.
- **Wait-Timed**. Wait for specified number of milliseconds (see Wait Interval).

Though the Wait Timeout can also be set through the collaboration, the advantage to setting it as a parameter is that the setting becomes the default. The default parameter value can still be overridden in the collaboration. This parameter is only in effect when the getWithOptions method (not the get method) is used.

Required Values

Select either Unlimited, No-Wait or Wait-Timed. No-Wait is the configured default.

Wait Interval

Description

Integer-set. Specifies the number of milliseconds to wait for a message to arrive on the queue when calling getWithOptions. This option only applies when the Wait-Timed option has been selected for the Wait Timeout. If this is left blank, and Wait-Timed is chosen, a value of 0 (zero) will be used.

Though the Wait Interval can also be set through the collaboration, the advantage to setting it as a parameter is that the setting becomes the default. The default parameter

value can still be overridden in the collaboration. This parameter is only in effect when the getWithOptions method (not the get method) is used.

Required Values

An integer in the range of 0 to 200000000. The configured default is 0.

Implementation (ETD)

This chapter contains basic information for implementing the ETD-based Java-enabled MQSeries e*Way in a production environment. A sample schema is included on the installation CD-ROM for the ETD-based implementation of the e*Way. In addition, examples are provided detailing how the various components of the sample schema were created. For more information on creating and configuring e*Way components see the *e*Gate Integrator User's Guide*.

An additional sample schema using the ETD-based implementation for **OS/390 V2R10** is also available at../samples/ewmq/MQSeriesETDSample_os390.zip on the installation CD-ROM. For more information on the OS/390 sample schema see **The MQSeries (ETD) Sample Schema for OS/390** on page 95

8.1 MQSeries (ETD) Sample Implementation Components

The Java ETD-based MQSeries e*Way is an *application specific* e*Way which allows e*Gate to connect with IBM's MQSeries applications. With the MQSeries e*Way installed along with the e*Gate Integrator, schema's can be created to receive, transform and route data through e*Gate in a predefined system configuration.

The following pages contain a sample implementation which serves to explain how the components for an ETD-based MQSeries sample schema are created. The Host and Control Broker are automatically created and configured during the e*Gate installation. The default name for each is the name of the host on which you are installing the e*Gate Enterprise Manager GUI. To complete the sample implementation of the ETD-based MQSeries e*Way will require the following:
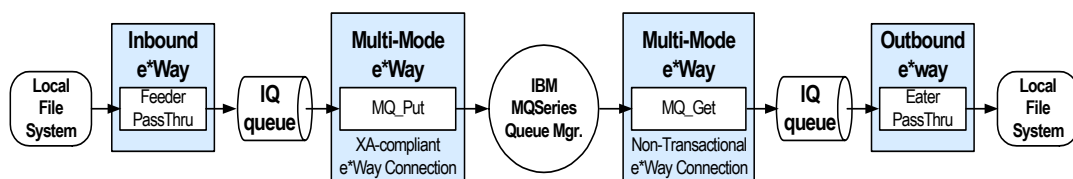
- **Install IBM's MQSeries** (See **External System Requirements** on page 10) and **MQSeries Queue Manager:** The MQSeries Server 5.2 and MQSeries Queue Manager are installed on the localhost. Also install **Java Classes for MQSeries 5.2** and the **MA88 SupportPac** from IBM.

- **Install the MQSeries e*Way:** The MQSeries e*Way is installed as an Add-on to the e*Gate Integrator. For directions on installing the MQSeries e*Way from the CD-ROM to your specific operating system, see **Installation** on page 12.

- **Create e*Ways:** e*Ways connect with external systems to poll or send data. They also transform and route data. Multi-Mode e*Ways are used to run Java Collaborations that utilize e*Way Connections to send and receive Events to and from multiple external systems.

- **Configure e*Way Connections:** An e*Way Connection is the encoding of access information for a specific external connection. The e*Way Connection configuration file contains the parameters necessary for communicating with IBM's MQSeries and specifying the MQSeries Queue Manager.

- **Create Event Types:** Each packet of data within e*Gate is referred to as an Event. Event Types are data labels that allow e*Gate to process and route specific Events differently. Data is not routed in e*Gate without an Event Type.

- **Create Intelligent Queues:** Non-volatile storage for data traveling through the e*Gate system is provided by creating Intelligent Queues (IQs). The IQ Manager oversees the activities of the individual storage locations. The exact behavior of each IQ is determined by the IQ Service configuration. The MQSeries e*Way uses the MQSeries IQ Service.

- **Create Collaboration Rules:** Collaboration Rules determine how input Event Types are modified to the format of specific output Event Types. A Collaboration Rule defines what type of data is received, how it will be transformed and what type of data will be published.

- **Create Collaborations:** A Collaboration is a message bus in e*Gate that specifies the name and source of the incoming Event Types, the Collaboration Rules that will be applied to the Event, and the name, destination and expiration date of the outgoing Event Types. A Collaboration designates the Subscriber, which "listens" for Events of a known type from a given source, and the Publisher, which distributes the transformed Event to a specified recipient.

- **Set the CLASSPATH Variable:** The Final Step in creating and configuring the MQSeries e*Way is to set the IBM MQSeries Java .jar files and the com.ibm.m9.jar files in the environment CLASSPATH variable.

8.1.1 The MQSeries (ETD) Sample Schema

The following sections describe how to define and associate each of the above components. This sample implementation will demonstrate how the Java-enabled MQSeries e*Way intercepts, stores, manipulates, and manages data in association with IBM MQSeries.

Figure 16 The MQSeries (ETD) Sample Implementation

8.1.2 Step One: Creating the IBM MQSeries Queue

Step one in creating the MQSeries e*Way is to install and configure **IBM MQSeries Server 5.2** and the **IBM MQSeries Queue Manager** on the localhost. It is also necessary to install **Java Classes for MQSeries** (See **External System Requirements** on page 10) and the **IBM MA88 SupportPac**.

It is assumed that the reader is experienced in the use of IBM MQSeries Queue Manager. For more information on IBM MQSeries Queue Manager please see MQSeries Queue Related Commands, Chapter 9, in the e*Gate Integrator Intelligent Queue Services Reference Guide. For the sample implementation do the following:

1 Open IBM MQSeries Explorer.

2 Create a new Queue Manager.

3 From the IBM MQSeries Queue Manager create a new queue.

*Note: Unlike the JMS-based MQSeries e*Way schema, the ETD-based schema does not require the MQSeries Queue name and the Event Type name to be the same.*

Regarding IBM MQSeries Server and Queue Manager Limits and Settings

- When using MQSeries Queue Manager on Unix, the user must be a member of the mqm group to create and start MQ Series Queue manager.

- It is essential that the MQSeries Administrator regularly monitor the number of messages in the queue. Message expiration settings should be set to allow for extended storage.

- MQseries is limited in the number of messages that can be sent before a commit is executed, and the number of physical messages that can exist on the queue at any one time. This can result in exception errors when upper limits for these numbers are exceeded. Memory and performance of the specific server may also effect the results.

Publishing Messages with MQSeriesJMS to a non-JMS conversant e*Way

The JMS standard specifies a header which includes encoding and reply information. This header is prepended to any message published by the IBM JMS classes. Non-JMS subscribers (that is, those using a non JMS API to MQSeries, such as the IBM C API) will not be able to separate the JMS header from the body. To remedy this, the user is advised to suppress the publication of the JMS header, if publishing to non-JMS subscribers, using the following mechanism.

To send messages to a non-JMS MQ (Monk MQSeries) e*Way, call send() manually from within the collaboration rules containing the following URI:

"queue://<QMGR_NAME>/<QNAME>?targetClient=1"

For example:

"queue://EMEO2T/QR.EME01?targetClient=1"

If this is not done then a 200+ byte header is pre-appended to the payload and placed in the MQ queue and could easily throw off the non JMS conversant MQ reader.

For further information see the IBM manual **MQSeries**, **Using Java**, Chapter 10, at:
http://www-4.ibm.com/software/ts/mqseries/library/manualsa/manuals/crosslatest.html

8.1.3 Step Two: Installing the MQSeries e*Way and Creating a New Schema

Step two is to install the MQSeries e*Way. For directions on installing the MQSeries e*Way on your specific operating system, see **Chapter 2, Installation, on page 12**.

Importing the Sample Schema

Once the MQSeries e*Way is installed, a new schema must be created. While it is possible to use the default schema, it is recommended that you create a separate schema for testing purposes. A sample schema, **MQSeriesETDSample.zip**, is included on the CD-ROM. To import the sample schema once the MQSeries e*Way is installed, do the following:

1 Start the e*Gate Enterprise Manager GUI.

2 When the Enterprise Manager prompts you to log in, select the host that you specified during installation and enter your password.

3 You will then be prompted to select a schema. Click on **New**.

4 Enter a name for the new Schema. In this case, for the sample, enter **MQSeriesETDSample**, or any name as desired.

5 Select **Create from export**. Click **Find** and navigate to following folder on the CD-ROM **\eGate\samples\ewmq** and select **MQSeriesETD.zip**.

6 Click open to import the sample schema.

The e*Gate Enterprise Manager opens to the new schema. The schema must be configured to match the specific system before it can run. See **Multi-Mode e*Way Configuration** on page 16 and **e*Way Connection Configuration (ETD)** on page 59 for directions on configuring the schema components for your system. Further information on importing and configuring the sample schema is available on the Readme file included with the sample

The following steps are included to demonstrate how the components of the sample schema are created.

8.1.4 Step Three–Creating and Configuring the e*Ways

Step three is to create the e*Ways. e*Ways are components used for transporting and transforming data. They always interface with at least one external system, and Multi-Mode e*Ways can use e*Way Connections to interface with many external systems. For the sample implementation four e*Ways are required.

- Inbound (Feeder)
- Outbound (Eater)

- Multi-Mode (MQ_Get)

- Multi-Mode (MQ_Put)

The following sections provide instructions for creating each e*Way.

Inbound e*Way (Feeder)

1 Select the Navigator's **Components** tab.

2 Open the host on which you want to create the e*Ways.

3 Select the **Control Broker** that will manage the new e*Ways.

4 On the palette, click the **Create a New e*Way** button.

5 Enter the name of the new e*Way (in this case, **Feeder**). Click **OK**.

6 Right-click the **Feeder** e*Way, and select **Properties** to edit its properties.

7 When the e*Way Properties window opens, click on the **Find** button beneath the **Executable File** field and select **stcewfile.exe** as the executable file.

Figure 17   Inbound e*Way Properties
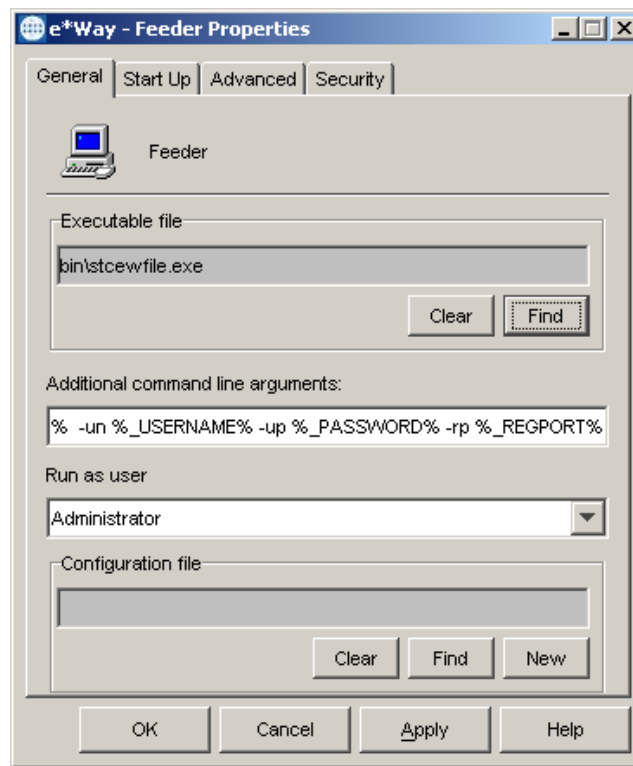
8 Under the **Configuration File** field, click on the **New** button. The **Edit Settings** dialog box opens. Set the following for this configuration file.

Table 5   Configuration Parameters for the Inbound e*Way

Parameter	Value
General Settings (unless otherwise stated, leave settings as default)	

Table 5   Configuration Parameters for the Inbound e*Way

Parameter	Value
AllowIncoming	YES
AllowOutgoing	NO
PerformanceTesting	NO
Outbound Settings	Default
Poller Inbound Settings	
PollDirectory	C:\INDATA
InputFileExtension	*.fin (input file extension)
PollMilliseconds	1000
Remove EOL	YES
MultipleRecordsPerFile	YES
MaxBytesPerLine	4096
BytesPerLineIsFixed	NO
File Records Per eGate Event	1
Performance Testing	Default

Note: *For information on configuring the specific parameters of the stcewfile e*Way see the*
Standard e*Way Intelligent Adapter User's Guide*.*

9   Use the Startup, Advanced, and Security tabs to modify the default settings for each
e*Way you configure.

D   Use the **Startup** tab to specify whether the e*Way starts automatically, or restarts
after abnormal termination or due to scheduling and so forth.

E   Use the **Advanced** tab to specify or view the activity and error logging levels as
well as the Event threshold information.

F   Use **Security** to view or set privilege assignments.

10   After selecting the desired parameters, save the configuration file (**Feeder.cfg**) and
Promote to Run Time.

11   Select **OK** to close the **e*Way Properties** window.

Outbound e*Way (Eater)

1   Repeat step 1-9 above to create the Outbound e*Way changing the name in steps 5
and 6 to **Eater**.

2   Replace the following parameters for those in step 8.

Table 6   Configuration Parameters for the Inbound e*Way

Parameter	Value
General Settings (unless otherwise stated, leave settings as default)	
AllowIncoming	NO

Table 6   Configuration Parameters for the Inbound e*Way

Parameter	Value
AllowOutgoing	YES
PerformanceTesting	NO
Poller Outbound Settings	
OutputDirectory	C:\DATA
OutputFileName	output%d.dat
MultipleRecordsPerFile	YES
MaxRecordsPerFile	10000
AddEOL	YES
Poller Inbound Settings	Default
Performance Testing	Default

3   After selecting the desired parameters, **Save** the configuration file (**Eater.cfg**) and **Promote to Run Time**.

4   Select **OK** to close the **e*Way Properties** window.

Multi-Mode e*Way (MQ_Get)

1   Select the **Navigator's Components** tab.

2   Open the host on which you want to create the e*Way.

3   Select the **Control Broker** that will manage the new e*Way.

4   On the palette, click the **Create a New e*Way** button.

5   Enter the name of the new e*Way (in this case, **MQ_Get**), then click **OK**.

6   Right-click the **MQ_Get** e*Way and select **Properties** to edit its properties.

7   When the e*Way Properties window opens, click the **Find** button beneath the **Executable File** field, and select **stceway.exe** as the executable file.

8   To configure the Multi-Mode e*Way's parameters, select **New** under the Configuration File field and enter the following parameters.

Table 7   Configuration Parameters for the MQ_Get MultiMode e*Way

Parameter	Value
JVM Settings (unless otherwise stated, leave settings as default)	
JNI DLL absolute pathname	C:\eGate\client\bin\Jre\jvm.dll (or absolute path to proper JNI DLL)
CLASSPATH Append From Environmental Variable	YES

See **"Multi-Mode e*Way Configuration" on page 16** for details on the parameters associated with the Multi-Mode e*Way.

9   In the **e*Way Properties** window, use the **Startup**, **Advanced**, and **Security** tabs to modify the default settings for each.

   G   Use the **Startup** tab to specify whether the e*Way starts automatically, restarts after abnormal termination or due to scheduling, etc.

   H   Use the **Advanced** tab to specify or view the activity and error logging levels, as well as the Event threshold information.

   I   Use **Security** to view or set privilege assignments.

10   After selecting the desired parameters, save the configuration file (**MQ_Get.cfg**) and **Promote to Run Time**.

11   Click **OK** to close **e*Way Properties** window.

Multi-Mode e*Way (MQ_Put)

1   Repeat step 1-9 above to create the MQ_Put Multi-Mode e*Way changing the name in steps 5 and 6 to **MQ_Put**.

2   After selecting the desired parameters, **Save** the configuration file (**MQ_Put.cfg**) and **Promote to Run Time**.

3   Click **OK** to close **e*Way Properties** window.

8.1.5   Step Four: Create the e*Way Connection

Step four is to create and configure the e*Way Connections. The e*Way Connection configuration file contains the settings necessary for communicating with IBM MQSeries and specifying the MQSeries Queue Manager. For this sample two e*Way Connections are created.

MQConn_Get e*Way Connection

1   Select the **e*Way Connection** folder on the **Components** tab of the e*Gate Navigator.

2   On the palette, click the **Create a New e*Way Connection** button.

3   Enter the name of the **e*Way Connection**, then click **OK.** (For the purpose of this sample, the e*Way Connection is defined as "**MQconn_Get**".)

4   Double-click the new **e*Way Connection** to edit its properties. The **e*Way Connection Properties** dialog box opens.

5 In the e*Way Connection Type field, select **MQSeriesETD** from the drop-down list box.

Figure 18   e*Way Connection Properties
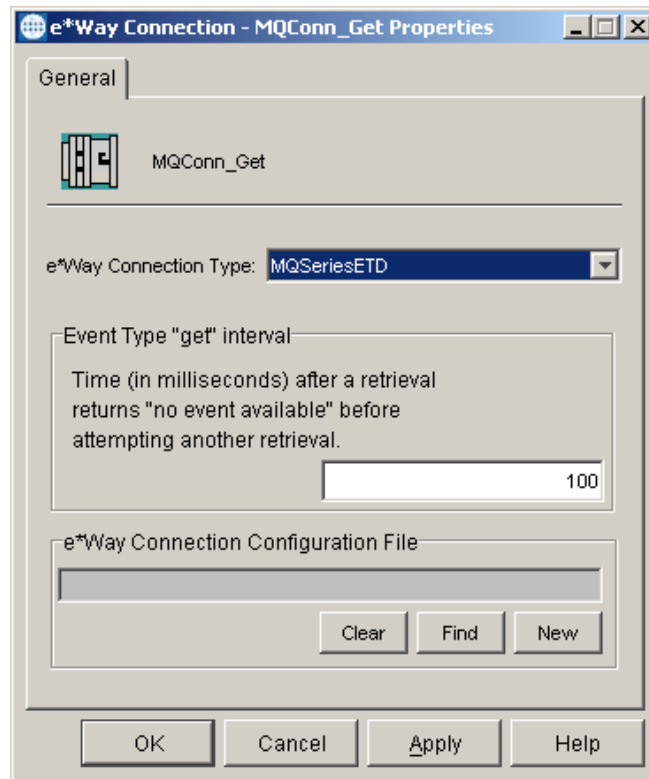
6 Enter the **Event Type "get" Interval** in the dialog box provided. 10000 milliseconds (or 10 seconds) is the configured default. The "get interval is the intervening period at which, when subscribed to, the e*Way connection is polled. For the purpose of this sample set the "get interval to 100.

7 To configure the e*Way Connection parameters, click the **New** button under the e*Way Connection Configuration File field.

8 The e*Way Connection editor opens, select the following parameters.

Table 8   Configuration Parameters for the MQConn_Get e*Way Connection

Parameter	Value
General Settings (unless otherwise stated, leave settings as default)	
Transaction Type	Non-Transactional
MQSeries	
Queue Manager Name	MQ_mgr (a valid queue manager)
Port Number	1414 (a valid port number)
connector	Default
Default GetMessageOptions	

Table 8   Configuration Parameters for the MQConn_Get e*Way Connection

Parameter	Value
Wait Timeout	Wait-Timed
Wait Interval	10000

For more information on the MQSeries e*Way Connection Type parameters, see **"e*Way Connection Configuration (ETD)" on page 59**.

9 Save the **MQConn_Get.cfg** file.

10 From the File menu select **Promote to Run Time** to move the file to the e*Way's run time environment.

11 Click **OK** to close the Properties dialog box.

MQConn_PutXA e*Way Connection

1 Repeat step 1-8 above to create the **MQConn_PutXA** e*Way Connection changing the name in step 3 to **MQConn_PutXA**.

2 Replace the following parameters for those in step 8.

Table 9   Configuration Parameters for the MQConn_PutXA e*Way Connection

Parameter	Value
General Settings (unless otherwise stated, leave settings as default)	
Transaction Type	XA-compliant
MQSeries	
Queue Manager Name	MQ_mgr (a valid queue manager)
Port Number	1415 (a valid port number)
Channel	Channel2Test
connector	Default
Default GetMessageOptions	
Wait Timeout	No-Wait
Wait Interval	0

3 Save the **MQConn_PutXA.cfg** file.

4 From the File menu select **Promote to Run Time** to move the file to the e*Way's run time environment.

5 Click **OK** to close the Properties dialog box.

8.1.6 Step Five: Creating Event Types

Step five is to create the Event Types. An Event Type is a class of Events with a common data structure. The e*Gate system packages data within Events and categorizes them into Event Types. What these Events have in common defines the Event Type and comprises the ETD.

Creating an Event Types Using the Standard ETD Wizard

The following procedures show how to create an ETD (Event Type Definition) using the Standard ETD Wizard.
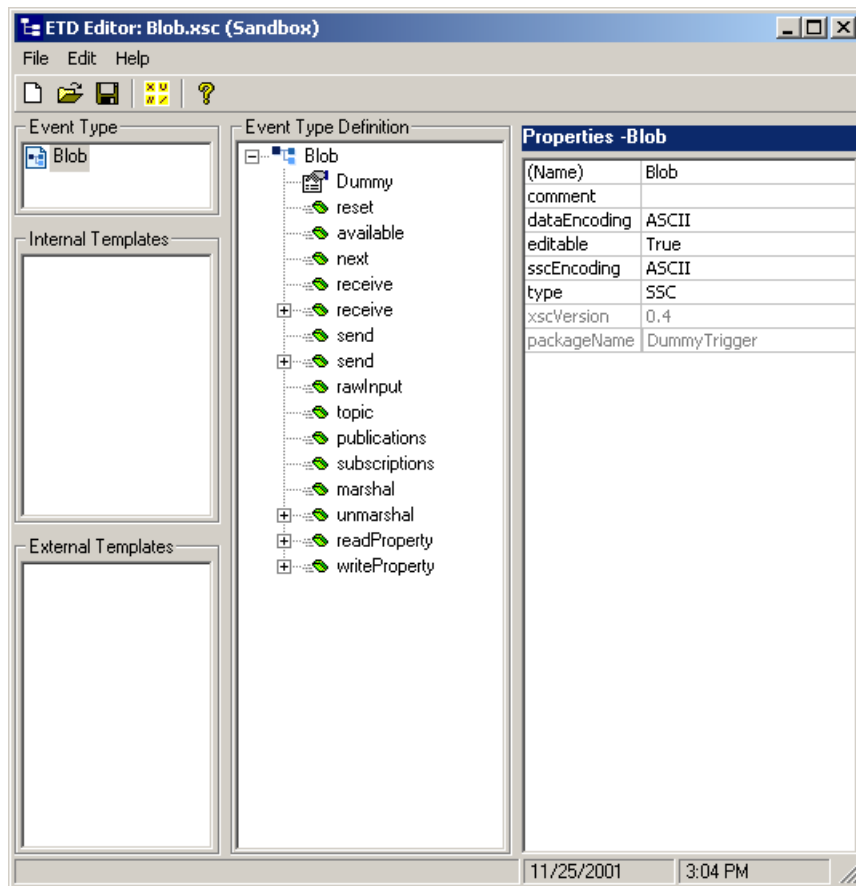
1 Highlight the **Event Types** folder on the **Components** tab of the e*Gate Navigator.

2 On the palette, click the **Create a New Event Type** button.

3 Enter the name of the **Event**, then click **OK.** For the purpose of this sample the Event Type is defined as **DummyTrigger**.

4 Double-click the new **Event Type** to edit its properties. The **Event Type Properties** dialog box opens.

5 Click the **New** button. The ETD Editor opens.

6 Select **New** from the File menu. The New Event Type Definition window opens.

Figure 19   Event Type Definition Wizards
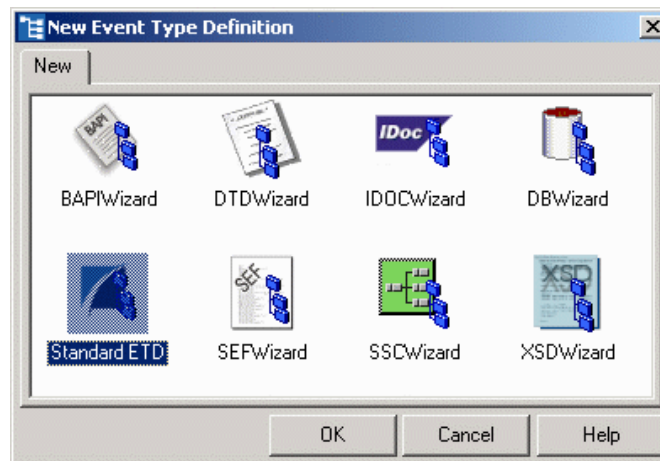
7 Select the **Standard ETD** wizard.

8 Enter the Root Node Name (for this case, "**Blob**").

9 Enter a package name where the ETD Editor can place all the generated Java classes associated with the created ETD. (For this sample, use **DummyTrigger** as the package name.) Click **Next** and **Finish** to close the wizard.

10 Right click **Blob** in the Event Type Definition pane of the ETD Editor, and select **Add Field, as Child Node**.

11 Triple-click on **Field1**, and rename it **Dummy**.

12 Select the **Dummy** node. The properties for the Dummy node are displayed in the Properties pane. Change the **endDelim** property to "**|**" (pipe).

Figure 20   Event Type Definition Editor

13   From the File menu, click **Compile and Save**. Save the .xsc file as
     DummyTrigger.xsc.

14   From the File menu, click **Promote to Run Time** to move the file to the run time
     environment.

15   Close the ETD Editor.

Creating Event Types From an Existing XSC

The following procedure shows how to create an Event Type Definition (ETD) from an
existing .xsc file.

1   Select the **Event Types** folder on the **Components** tab of the e*Gate Navigator.

2   On the palette, click the **Create a New Event Type** button.

3   Enter the name of the **Event Type** in the **New Event Type Component** window,
    then click **OK**. (For this sample, the Event Type is defined as "**fromMQ**.")

4   Double-click the new **Event Type** to edit its properties. The **Event Type Properties**
    dialog box opens.

5   Click the **Find** button under the **Event Type Definition** field.

6   Browse to and select **DummyTrigger.xsc**.

7   Click **OK** to close the Event Type Properties dialog box.

8.1.7 Step Six: Intelligent Queues

Step Six in configuring the MQSeries e*Way is to create the Intelligent Queues (IQs). IQs manage the exchange of information between components within the e*Gate system, providing non-volatile storage for data as it passes from one component to another. IQs use IQ Services to transport data. IQ Services provide the mechanism for moving Events between IQs, handling the low-level implementation of data exchange (such as system calls to initialize or reorganize a database).

To create and modify an Intelligent Queue for the MQSeries e*Way

1   Select the Navigator's **Components** tab.

2   Open the host on which you want to create the IQ.

3   Open a **Control Broker**.

4   Select an **IQ Manager**.

5   On the palette, click the **Create a New IQ** button.

6   Enter the name of the new IQ, then click **OK.** (For this case, **queue**.)

7   Double-click the new **IQ** to edit its properties.

8   On the **General** tab, specify the **Service** (for this sample, **STC_Standard**). The **STC_Standard** IQ Service provides sufficient functionality for most applications. If specialized services are required, custom IQ Service DLLs may be created.

9   Specify and the **Event Type Get Interval**. The default **Event Type Get Interval** of 100 Milliseconds is satisfactory for the purposes of this initial implementation.

10 On the **Advanced** tab, make sure that **Simple publish/subscribe** is checked under the **IQ behavior** section.

11 Click OK to close the **IQ Properties** window

8.1.8 Step Seven: Collaboration Rules

Step seven in creating the ETD-based MQSeries e*Way is to create the Collaboration Rules that will extract and process selected information from the source Event Type defined earlier, according to its associated Collaboration Service. The **Default Editor** can be set to either **Monk** or **Java**. From the **Enterprise Manager Task Bar,** select **Options** and click **Default Editor**. Make sure that the default is set to **Java**.

Creating Pass Through Collaboration Rules

1   Select the Navigator's **Components** tab in the e*Gate Enterprise Manager.

2   In the **Navigator**, select the **Collaboration Rules** folder.

3   On the palette, click the **Create New Collaboration Rules** button.

4 Enter the name of the new Collaboration Rule Component, then click **OK** (in this case, use **crDataIn**).

5 Double-click the new Collaboration Rules Component. The **Collaboration Rules Properties** window opens.

6 Select **Pass Through** from the drop-down box for the Service field.

Figure 21 Pass Through Collaboration Properties
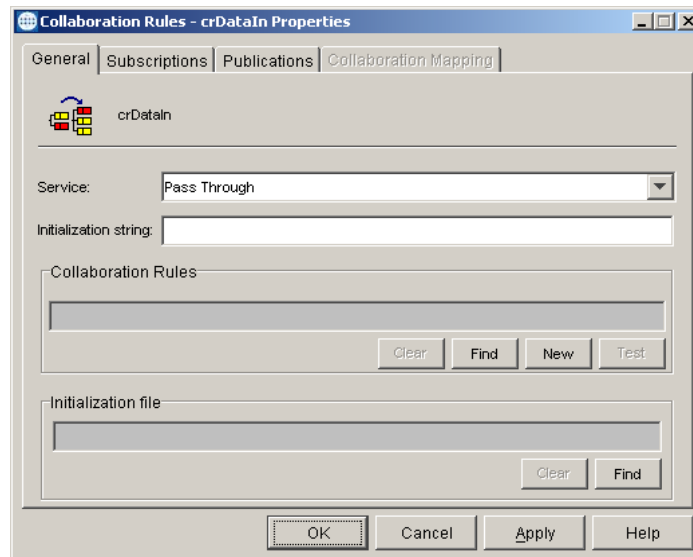
7 Go to the **Subscriptions** tab. Select **DummyTrigger** under **Available Input Event Types**, and click the right arrow to move it to **Selected Input Event Types**. The box under **Triggering Event** should be checked.

Figure 22 Pass Through Collaboration Properties, Subscriptions Tab
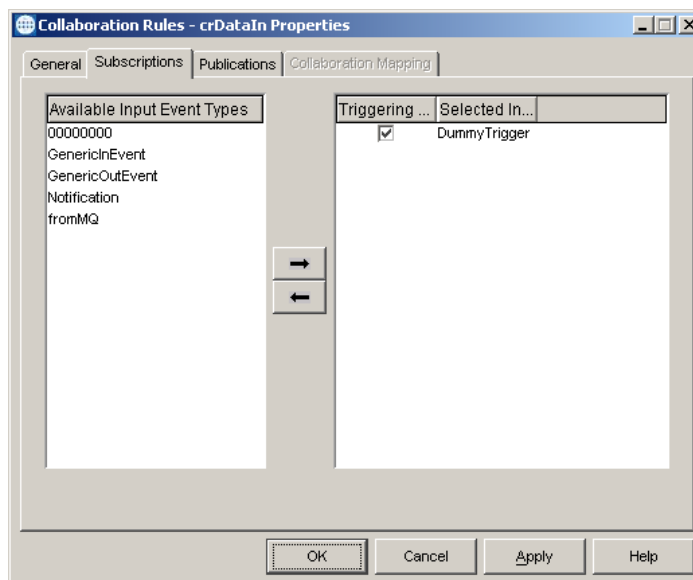
8 Go to the **Publications** tab. Select **DummyTrigger** under **Available Output Event Types**, and click the right arrow to move it to **Selected Output Event Types**. The Radio button under **Default** will be enabled.

9 Click **OK** to close the **Collaboration Rules, Pass Properties** window.

10 Repeat steps 1-9 above to create the **crDataOut** Collaboration Rules, changing the name in step 4 to **crDataOut** and the selected Input and Output Event Types in steps 7 and 8 to **fromMQ.**

Creating Java Collaboration Rules

For the purpose of the sample schema two Java Collaboration Rules files are created.

Creating MQCollab Collaboration Rules

1 Select the Navigator's **Components** tab in the e*Gate Enterprise Manager.

2 In the **Navigator**, select the **Collaboration Rules** folder.

3 On the palette, click the **Create New Collaboration Rules** button.

4 Enter the name of the new Collaboration Rule, then click **OK** (for this case, use **MQCollab**).

5 Double-click the new Collaboration Rules Component to edit its properties. The **Collaboration Rules Properties** window opens.

6 The **Service** field defaults to **Java**. The **Collaboration Mapping** tab is enabled, and the **Subscriptions** and **Publications** tabs are disabled.

7 In the **Initialization string** field, enter any required initialization string for the Collaboration.

Figure 23   Collaboration Rules - Properties
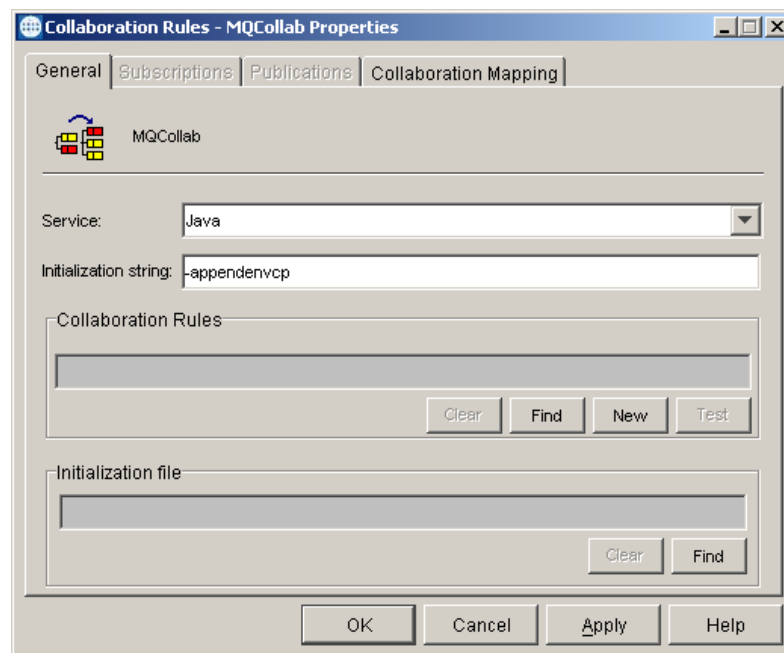
8 Select the **Collaboration Mapping** tab.

9 Using the **Add Instance** button, create instances to coincide with the Event Types.

For this sample, do the following:

10 In the **Instance Name** column, enter **outbound** for the instance name.

11 Click **Find**, navigate to **etd\MQSeriesETD.xsc**, double-click to select. **MQSeriesETD.xsc** is added to the **ETD** column of the instance row.

12 In the **Mode** column, select **Out** from the drop–down menu available.

13 The **Trigger** setting defaults to **N/A**.

14 The **Manual Publish** setting is clear.

15 Repeat steps 9–13 using the following values:

   ◆ Instance Name: **trigger**

   ◆ ETD: **DummyTrigger.xsc**

   ◆ Mode: **In**

   ◆ Trigger: **select**

   ◆ Manual Publish: **N/A**

Note: *At least one of the ETD instances used by the Collaboration must be checked as the trigger.*

For specific information on creating and configuring Collaboration Rules, see the e*Gate Integrator User's Guide.

Figure 24 Collaboration Rules - Collaboration Mapping Properties
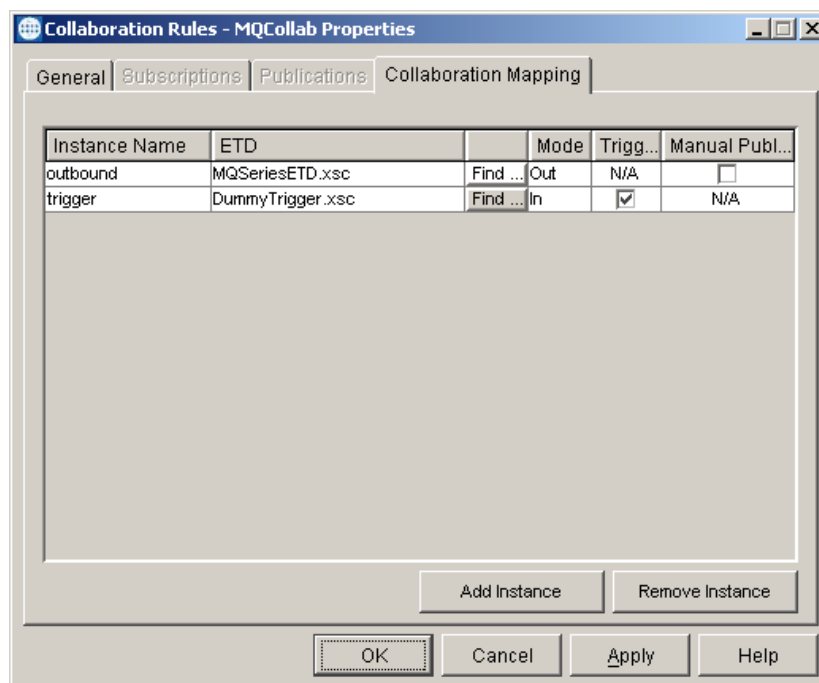
16 Select the **General** tab. Under the Collaboration Rule field, select **New**. The **Collaboration Rules Editor** opens.

17 Expand to full size for optimum viewing, expanding the Source and Destination Events as well.

8.1.9 Using the Collaboration Rules Editor

Part two of step seven is to define the business logic using the Collaboration Rules Editor. The Java Collaboration Rules Editor is the GUI used to create and modify Java Collaboration Rules. A Java Collaboration Rule is created by designating one or more source Events and one or more destination Events and then setting up rules governing the relationship between fields in the Event instances.

Note: *In order to compile the Collaboration that uses MQSeriesETD.xsc, first add com.ibm.mq.jar to the User Classpath from Tools, Options in the Collaboration Rules editor.*

1 Highlight **retBoolean** in the **Business Rules** pane.

All of the user–defined business rules are added as part of this method.

2 Click the **rule** button on the Business Rules tool bar. A rule expression is added to the business rules.

3 From the Destination Events pane, drag-and-drop **MQOO_OUTPUT** into the Business Rules pane, Rules window. When prompted for *type of function for this node* select **set**. Place the curser between the last set of parentheses and type **true**.

Figure 25 Collaboration Rules Editor - Drag and Drop
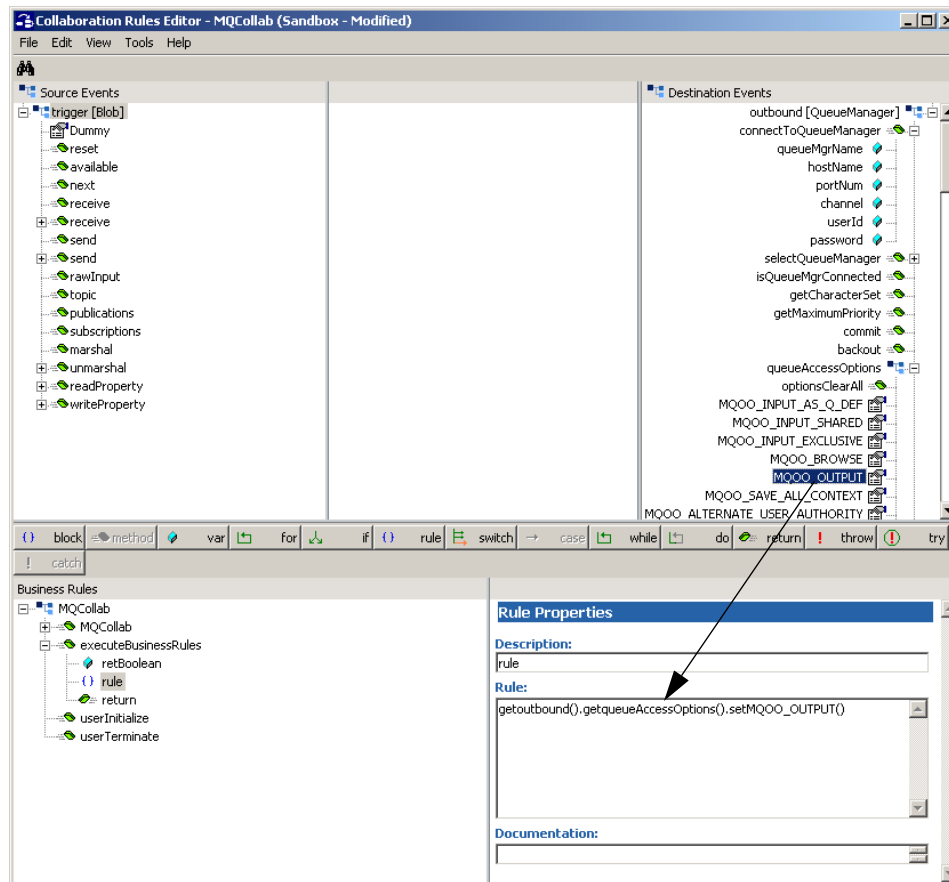

4 In the Description field type **set the queue access options**. This description now displays as the rule tag in the Business Rules pane.

5 Click the **rule** button on the Business Rules tool bar again. A rule expression is added to the Business Rules pane under the previous rule.

6 From the Destination Events pane, drag-and-drop the **accessQueue** method into the Business Rules pane, Rules window. When prompted for the **queueName** type in the name of the queue to which you have access.

Figure 26   Collaboration Rules Editor – accessQueue
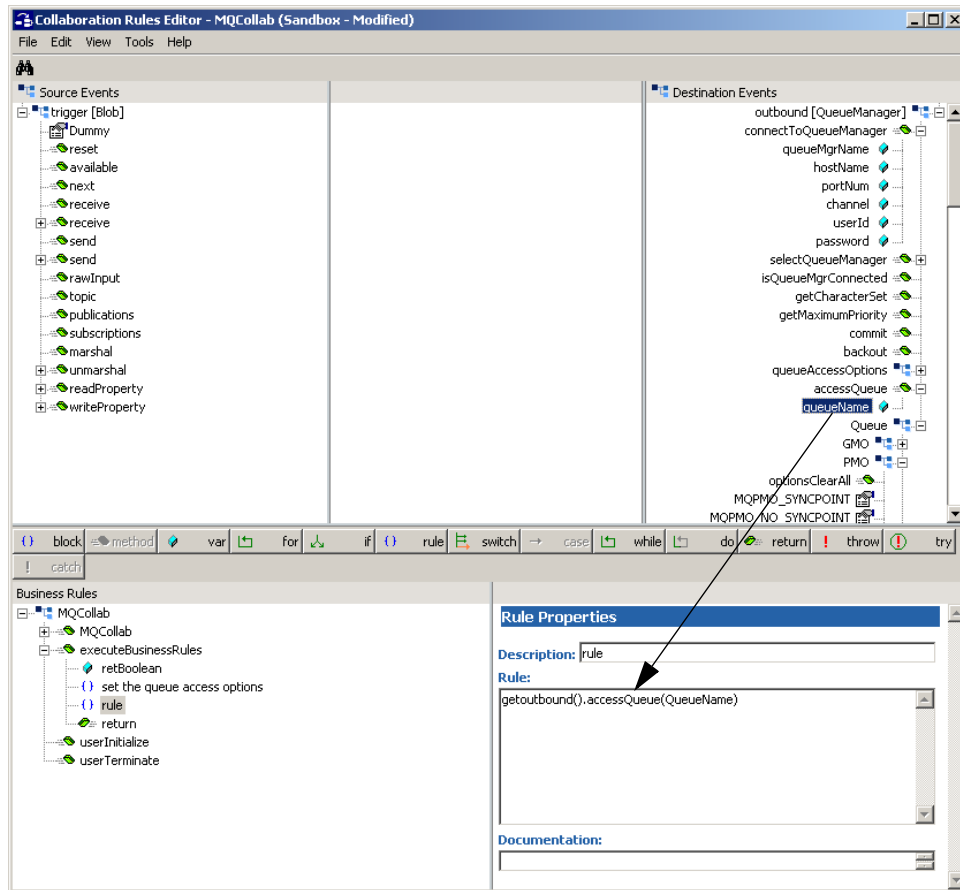
7   In the Description field type **access the queue**. This description now displays as the rule tag in the Business Rules pane.

8   Click the **rule** button to add another rule expression.

9   From the Destination Events pane, drag-and-drop **MQPMO_NO_SYNCPOINT** under PMO into the Business Rules pane, Rules window. When prompted for *type of function to insert for this node* select **set**. Place the curser between the last set of parentheses and type **true**.

10   In the Description field type **set the PutMessageOptions**. This description now displays as the rule tag in the Business Rules pane.

11   Click the **rule** button to add another rule expression.

12   From the Destination Events pane, drag-and-drop the **writeString** method under Message, MsgBody into the Business Rules pane, Rules window. When prompted for *a stringValue* type your message into the field. The value (message) can also be dragged and dropped to the **write** calls as appropriate, or to the MsgBody data node and call **writeData**.

Figure 27 Collaboration Rules Editor – writeString
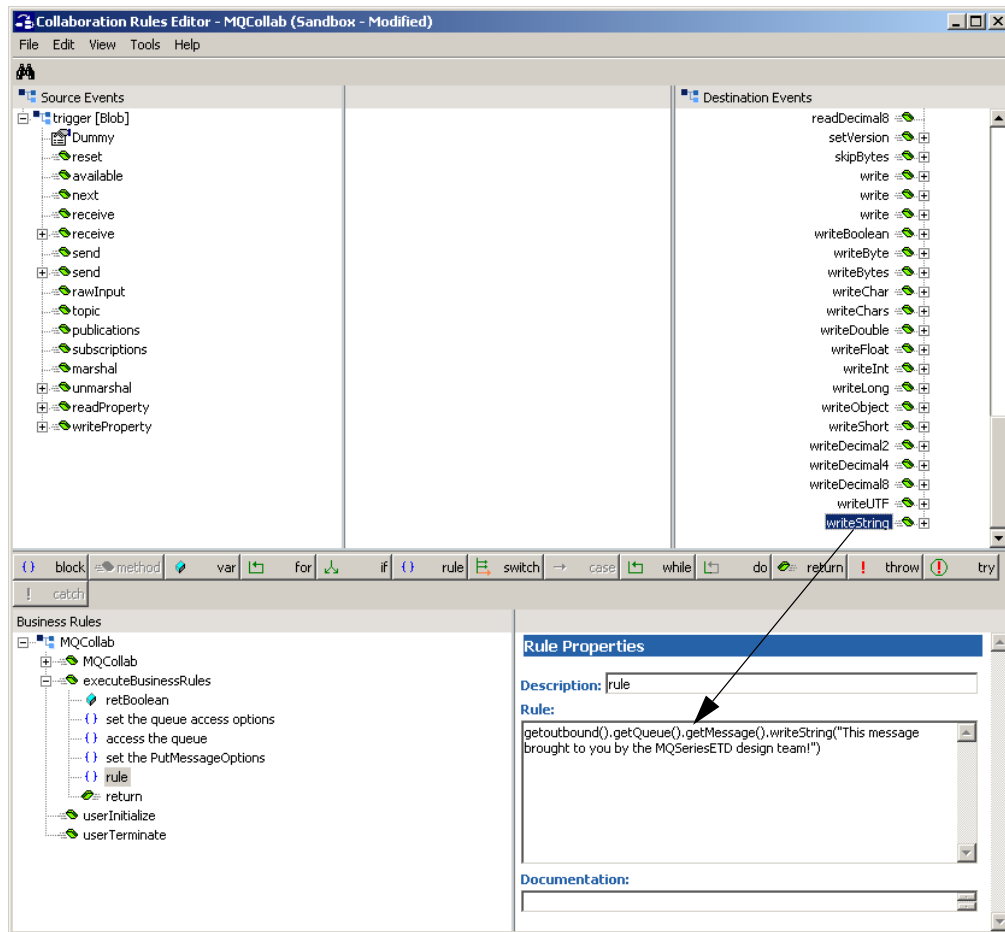
13 In the Description field type **write a string to the message**.

14 Once more, click the **rule** button to add another rule expression.

15 From the Destination Events pane, drag-and-drop the **putWithOptions** method into the Business Rules pane, Rules window.

16 In the Description field type **put the message on the queue**.

17 When all the business logic has been defined, the code can be compiled by selecting **Compile** from the **File** menu. In order to compile the Collaboration that uses MQSeriesETD.xsc, first add com.ibm.mq.jar to the User Classpath from Tools, Options in the Collaboration Rules editor. The **Save** menu opens, provide a name for the **.xpr** file. For the sample, use **MQCollab.xpr**. If the code compiles successfully, the message **Compile Completed** appears. If the outcome is unsuccessful, a Java Compiler error message appears. Once the compilation is complete, save the file and exit the Collaboration Rules Editor.

Figure 28   Collaboration Rules — Collaboration Rules Editor
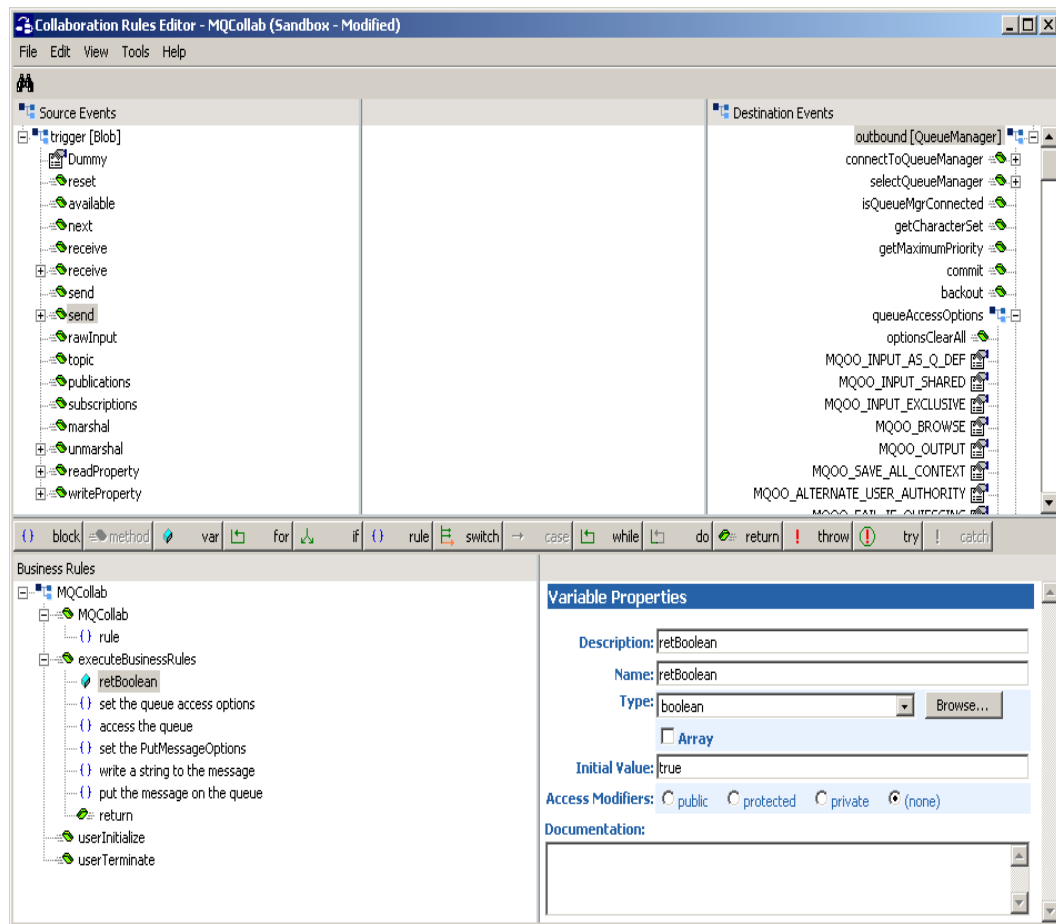
18   Under the **Collaboration Rules** field in the Collaboration Rules Properties dialog box, the path for the created .class file appears.

19   Under the **Initialization file** field, the path for the created **.ctl** file appears.

20   Click **OK** to close the **Properties** dialog box.

*Note: For detailed information on creating Collaboration Rules using the Java Collaboration Rules Editor see the e*Gate Integrator User's Guide.*

Creating the MQGetCollab Collaboration Rules

1   Click on the **Create New Collaboration Rules** button again. Name the new Collaboration Rules **MQGetCollab** for this sample.

2   Double-click the new Collaboration Rules. The Services field defaults to Java. Enter any required initialization string in the **Initialization string** field.

3   Select the **Collaboration Mapping** tab and create an instance and settings for the following;

   ◆ Instance Name — **inbound_MQ**

   ◆ ETD — **MQSeriesETD.xsc**

- Mode — **In**
- Trigger – **select**
- Manual Publish — **N/A**

4 Create another instance and settings for the following:

- Instance Name — **out**
- ETD — **DummyTrigger.xsc**
- Mode — **Out**
- Trigger – **N/A**
- Manual Publish — **Clear**

**Figure 29** Collaboration Rules - Collaboration Mapping Properties
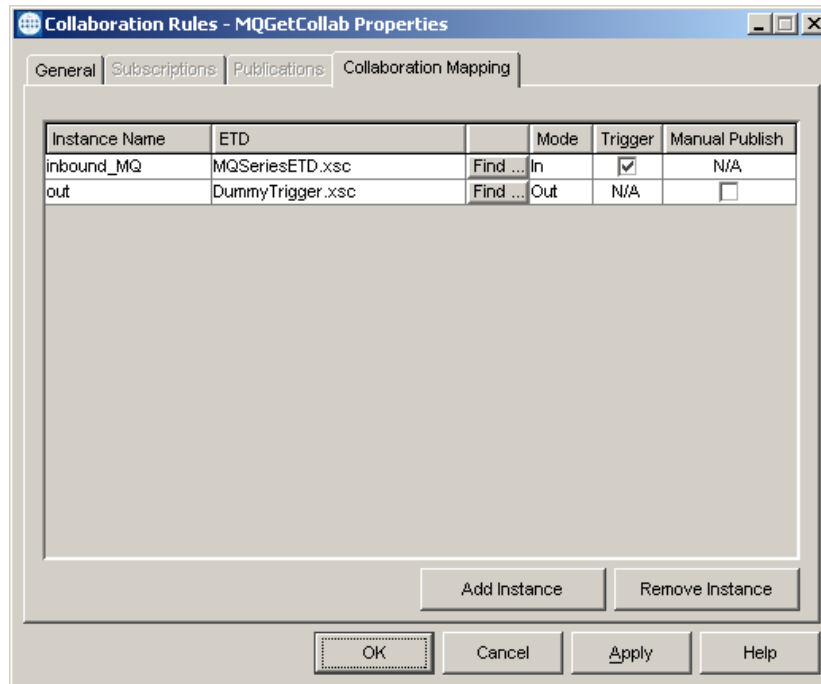
5 Select the **General** tab. Under the Collaboration Rule field, select **New**. The **Collaboration Rules Editor** opens. Expand to full size for optimum viewing, expanding the Source and Destination Events as well.

6 Highlight **retBoolean** in the **Business Rules** pane.

7 Click the **rule** button on the Business Rules tool bar. A rule expression is added to the business rules.

8 From the Source Events pane, drag-and-drop **MQOO_INPUT_AS_Q_DEF** into the Business Rules pane, Rules window. When prompted for *type of function for this node* select **set**. Place the curser between the last set of parentheses and type **true**.

9 In the Description field type **set the queue access options**. This description now displays as the rule tag in the Business Rules pane.

10 Click the **rule** button on the Business Rules tool bar again. A rule expression is added to the Business Rules pane under the previous rule.

11 From the Source Events pane, drag-and-drop the **accessQueue** method into the Business Rules pane, Rules window. When prompted for the **queueName** type in the name of the queue to which you have access.

12 In the Description field type **access the queue**. This description now displays as the rule tag in the Business Rules pane.

13 Click the **rule** button to add another rule expression.

14 From the Source Events pane, drag-and-drop **MQPMO_NO_SYNCPOINT** under PMO into the Business Rules pane, Rules window. When prompted for *type of function to insert for this node* select **set**. Place the curser between the last set of parentheses and type **true**.

15 In the Description field type **set the PutMessageOptions if any**. This description now displays as the rule tag in the Business Rules pane.

16 Click the **rule** button to add another rule expression.

17 From the Source Events pane, drag-and-drop the **getWithOptions** method into the Business Rules pane, Rules window.

18 In the Description field type **Calling getWithOptions()**.

19 Click the **rule** button to add another rule expression.

20 From the Source Events pane, drag-and-drop the **readData** method under Message, MsgBody into the Business Rules pane, Rules window.

21 In the Description field type **Calling readData()**.

22 From the Source Events pane, drag-and-drop the **Data** method under Message, MsgBody to **Dummy** in the Destination Events pane. A line appears between **Data** and **Dummy**, and the created code appears in the Rule Properties, Rules window. If necessary, edit the code in the Rule Properties, Rules window to appear as follows:

```
getout().setDummy(newString(getinbound_MQ().getQueue()
.getMessage().getMsgBody().getData()))
```

Figure 30 Collaboration Rules — Collaboration Rules Editor
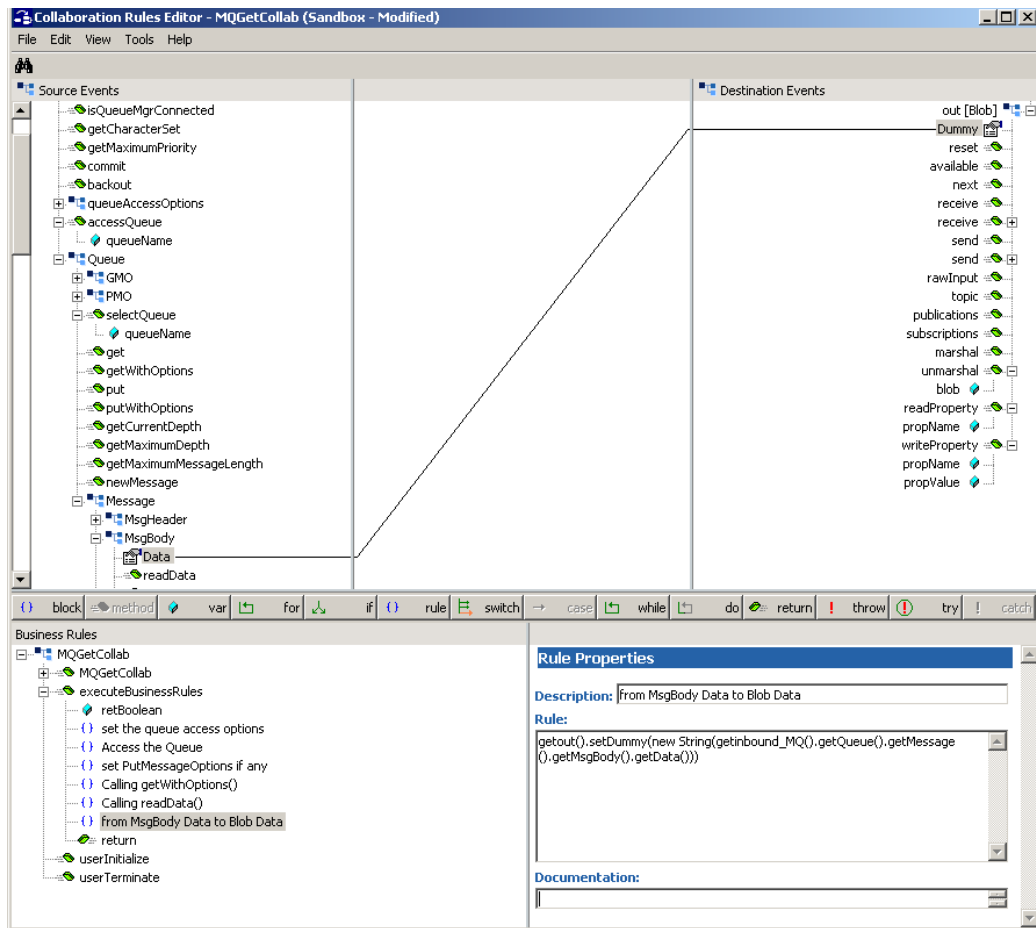
23 When all the business logic has been defined, the code can be compiled by selecting **Compile** from the **File** menu. In order to compile the Collaboration that uses MQSeriesETD.xsc, first add com.ibm.mq.jar to the User Classpath from Tools, Options in the Collaboration Rules editor. The **Save** menu opens, provide a name for the **.xpr** file. For the sample, use **MQGetCollab.xpr**. If the code compiles successfully, the message **Compile Completed** appears. If the outcome is unsuccessful, a Java Compiler error message appears. Once the compilation is complete, save the file and exit the Collaboration Rules Editor.

24 Click **OK** to close the **Properties** dialog box.

Note: *For detailed information on creating Collaboration Rules using the Java Collaboration Rules Editor see the e*Gate Integrator User's Guide.*

8.1.10 Step Eight–Collaborations

Step eight in creating the ETD-based MQSeries e*Way is to create the Collaborations. Collaborations are the components that receive and process Event Types, then forward the output to other e*Gate components or an external component. Collaborations consist of the Subscriber, which "listens" for Events of a known type (sometimes from a

given source), and the Publisher, which distributes the transformed Event to a specified recipient.

Creating the Inbound_eWay Collaboration

1 Create another new Collaboration In the e*Gate Enterprise Manager, select the Navigator's **Components** tab.

2 Open the host on which you want to create the Collaboration.

3 Select a **Control Broker.**

4 Select the **Feeder** e*Way to assign the Collaboration.

5 On the palette, click the **Create New Collaboration** button.

6 Enter the name of the new Collaboration, then click **OK.** (For the sample, "**FeederCollab**".)

7 Double-click the new **Collaboration** to edit its properties.

8 From the **Collaboration Rules** list, select the **Collaboration Rules** file that you created previously. (For the sample, "**crDataIn**".)

9 In the **Subscriptions** area, click **Add** to define the input Event Types to which this Collaboration will subscribe.

   A From the **Event Type** list, select the **Event Type** that you previously defined **DummyTrigger**.

   B Select the **Source** from the **Source** list. In this case, it should be **<External>**.

10 In the **Publications** area, click **Add** to define the output **Event Types** that this Collaboration will publish.

   A From the **Event Types** list, select the **Event Type** that you previously defined **DummyTrigger**.

   B Select the publication **Destination** from the **Destination** list. In this case, it should be **queue**.

   C The Priority column will default to **5**.

Figure 31 Inbound e*Way Collaboration Properties
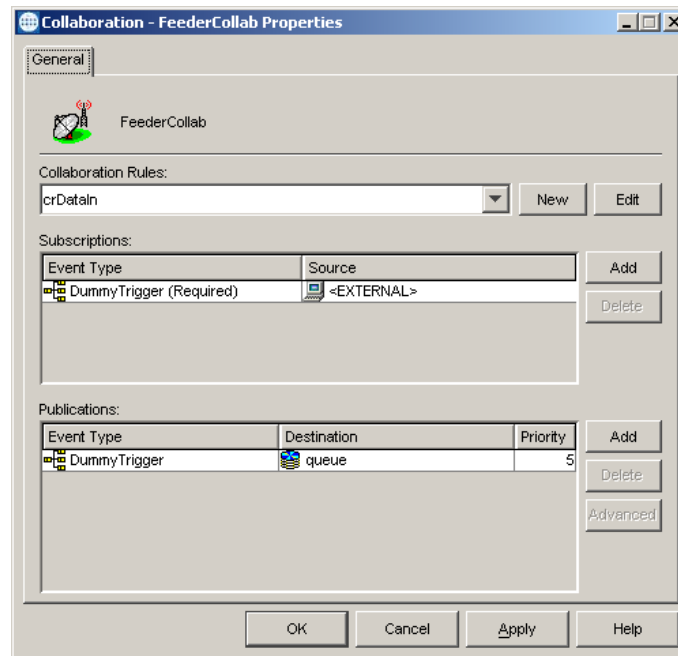
11 Click **OK** to close the **Collaboration Properties** window.

Creating the Multi Mode e*Way Collaboration

Two Collaboration will be created for the Multi-Mode e*Way **MQGetCollab**, and **colMQPutCollab**.

Creating the MQGetCollab Collaboration

1 To create the **MQGetCollab** Collaboration, Select the **MQ_Get** e*Way.

2 On the palette, click the **Create a New Collaboration** button.

3 Enter the name of the new Collaboration, then click **OK.** (For this sample, "**MQGetCollab**".)

4 Double-click the new Collaboration to edit its properties. The **Collaboration Properties** dialog box opens.

5 From the **Collaboration Rules** drop-down list box, select the Collaboration Rules file that you created previously. For the sample use **MQGetCollab.**

6 In the **Subscriptions** field, click **Add** to define the input Event Types to which this Collaboration will subscribe.

    A From the **Instance Name** list, select the **Instance Name** that you previously defined **inbound_MQ**.

    B From the **Event Type** list, select the **Event Type** previously defined **GenericOutEvent**.

    C Select the **Source** from the **Source** list. In this case, it should be **MQConn_Get**.

7 In the **Publications** area, click **Add** to define the output **Event Types** that this Collaboration will publish.

   A From the **Instance Name** list, select the **Instance Name** previously defined **out**.

   B From the **Event Types** list, select the **Event Type** named **GenericOutEvent**.

   C Select the **Destination** from the **Destination** list. In this case, it should be **Queue**.

   D The Priority column will default to **5**.

Figure 32 Collaboration Properties - MQGetCollab
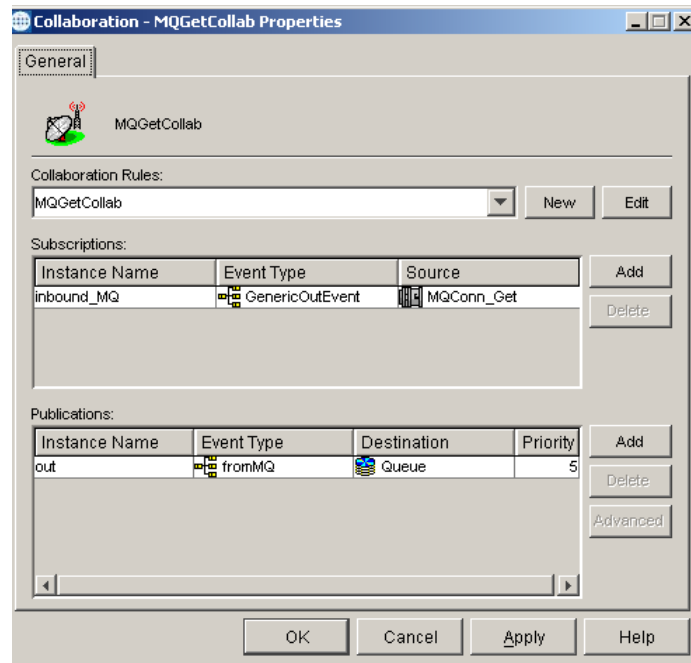
8 Click **OK** to close the Properties window.

Creating the MQPutCollab Collaboration

1 To create the **MQPutCollab**, repeat steps 1-6 above substituting the Collaboration name to **MQPutCollab**.

2 In the **Subscriptions** field, click **Add** to define the input Event Types to which this Collaboration will subscribe.

   A From the **Instance Name** list, select the **Instance Name** that you previously defined **trigger**.

   B From the **Event Type** list, select the **Event Type** previously defined **DummyTrigger**.

   C Select the **Source** from the **Source** list. In this case, it should be **FeederCollab**.

3 In the **Publications** area, click **Add** to define the output **Event Types** that this Collaboration will publish.

   A From the **Instance Name** list, select the **Instance Name** previously defined **outbound**.

B From the **Event Types** list, select the **Event Type** named **GenericOutEvent**.

C Select the **Destination** from the **Destination** list. In this case, it should be **MQConn_PutXA**.

4 The Priority column will default to **5**.

5 Click **OK** to close the Properties window.

8.1.11 Step Nine: Setting CLASSPATH Variable

The final step in creating and configuring the MQSeries e*Way is to set the IBM MQSeries Java .jar files in the environment CLASSPATH variable. This includes the following .jar files.

\MQSeries\Java\lib

\MQSeries\Java\lib\com.ibm.mq.jar

Also set the \MQSeries\Java\lib in your PATH.

For Unix, include /MQSeries/Java/lib in the library path as follows:

- Solaris: LD_LIBRARY_PATH

- HP-UX: SHLIB_PATH

- AIX: LIBPATH

If the CLASSPATH and PATH already exist, add the .jar files to the existing PATH and CLASSPATH.

Setting CLASSPATH variable on Windows

To set the .jar files from java classes in classpath do the following:

1 Right-click **My Computer** and select **Properties**. The System Properties window opens.

2 Select the **Advanced** tab and Click on **Environment Variables**. The Environment Variables window opens.

3 Under System Variables click the **New** button.

4 In the New System Variable window type **ClassPath** in the **Variable Name** field. In the Variable Value field type the absolute path for the first .jar file (See figure 12), and click **OK**.

Figure 33 Setting Environment Variables
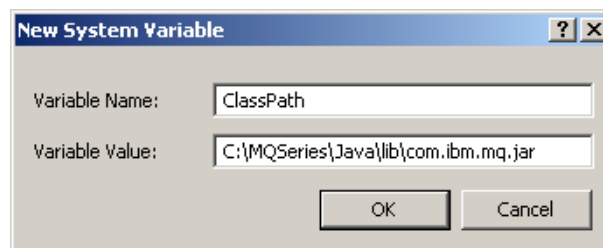
5 Repeat steps 3 and 4 for each of the MQSeries .jar files.

6 Under System Variables click the **New** button.

7 In the New System Variable window type **Path** in the **Variable Name** field. In the Variable Value field type the absolute path for \MQSeries\Java\lib and click **OK**.

8 Click **OK** to close the Environment Variables window and the System Properties window.

8.1.12 Execute the Schema

To execute the schema, do the following:

1 Go to the command line prompt, and enter the following:

```
stccb -rh hostname -rs schemaname -un username -up user password
-ln hostname_cb
```

Substitute *hostname*, schemaname, *username* and *user password* as appropriate.

2 Exit from the command line prompt, and start the e*Gate Monitor GUI.

3 When prompted, specify the *hostname* which contains the Control Broker you started in Step 1 above.

4 Select the schema.

5 After you verify that the Control Broker is connected (the message in the Control tab of the console will indicate command *succeeded* and status as *up*), highlight the IQ Manager, *hostname*_igmgr, then right-click and select **Start**.

6 Highlight each of the e*Ways, right-click the mouse, and select **Start**.

8.1.13 Error Messages

If there is an error, such as a failed connection, an exception is thrown by the module and logged to the error log file at egate/client/logs. The error log will appear similar to the following:

```
11:59:34.091 EWY I 11 (initialize.cxx:1035): Exception thrown: Failed to access queue:
MQRC_UNKNOWN_OBJECT_NAMEc
om.ibm.mq.MQException: MQJE001: Completion Code 2, Reason 2085
    at com.ibm.mq.MQQueueManager.accessQueue(MQQueueManager.java:1151)
    at com.ibm.mq.MQQueueManager.accessQueue(MQQueueManager.java:1196)
    at com.stc.eways.MQSeriesETD.MQSeriesConnector.accessQueue(MQSeriesConnector.java:395)
    at com.stc.eways.MQSeriesETD.MQSeriesETD.accessQueue(MQSeriesETD.java:291)
    at MQ_EMECollab.executeBusinessRules(MQ_EMECollab.java:106)
    at com.stc.jcsre.JCollaboration.translate(JCollaboration.java:97)
    at com.stc.common.collabService.JCCollabControllerImpl.
translate(JCCollabControllerImpl.java:1096
```

The reason code parameter or MQRC, in this case Reason 2085, appears in the first few lines of the error log. This reason code can be used in conjunction with IBMs online document, **MQSeries Messages**, Chapter 9 at:

**http://www-903.ibm.com/board/attach_files/mqseries/k1005706457257_messages.pdf**

The chapter lists reason codes, exceptions, the associated errors and the corrective actions to take. For the above example, the MQRC appears as follows:

2085	X'0825'	MQRC_UNKNOWN_OBJECT_NAME An MQOPEN or MQPUT1 call was issued, but the object identified by the ObjectName and ObjectQMgrName fields in the object descriptor MQOD cannot be found. One of the following applies: ▪ The ObjectQMgrName field is one of the following: ⬥ Blank ⬥ The name of the local queue manager ⬥ The name of a local definition of a remote queue (a queue-manager alias) in which the RemoteQMgrName attribute is the name of the local queue manager but no object with the specified ObjectName and ObjectType exists on the local queue manager. ▪ The object being opened is a cluster queue that is hosted on a remote queue manager, but local queue manager does not have a defined route to the remote queue manager. ▪ The object being opened is a queue definition that has QSGDISP(GROUP). Such definitions cannot be used with the MQOPEN and MQPUT1 calls. Corrective action: Specify a valid object name. Ensure that the name is padded to the right with blanks if necessary. If this is correct, check the queue definitions.

8.2   The MQSeries (ETD) Sample Schema for OS/390

An additional sample schema using the ETD-based implementation is available for **OS/390 V2R10**. The sample, on the installation CD-ROM at ../samples/ewmq/MQSeriesETDSample_os390.zip., is similar in configuration and operation to the MQSeriesETDSample described in this chapter with the addition of the outgoing encoder. All Data within a Java Collaboration is generally encoded in the ISO-8859-1 code page. The Outgoing Encoder is necessary in the OS/390 environment to produce EBCDIC output ("cp037").

Figure 34   MQGetCollab Collaboration Rule with the Outgoing Encoder
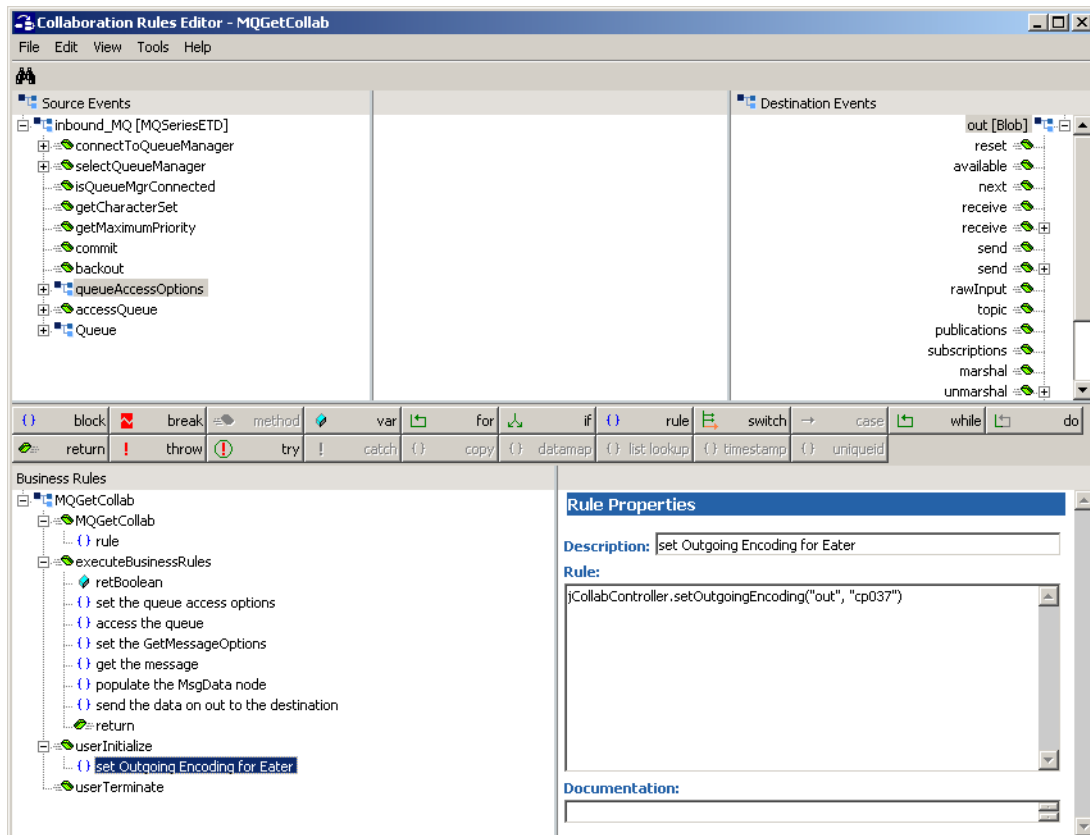
To use the MQSeries (ETD) Sample Schema for OS/390 import the sample, MQSeriesETDSample_os390.zip., and configure the components using the directions included in this chapter for the MQSeries ETD sample schema.

Java Methods (ETD)

The MQSeries e*Way contains Java methods that are used to extend the functionality of the e*Way. These methods are contained in the following classes:

- MQSeriesETD Class
- GMO Class (GetMessageOptions)
- PMO Class (PutMessageOptions)
- Message Class

9.1   MQSeriesETD Class

MQSeriesETD class methods are located under the **QueueManager node** and **Queue node.**

The MQSeriesETD class is defined as:

```
public class MQSeriesETD
```

The MQSeriesETD class extends **com.stc.jcsre.SimpleETDImpl** and implements **com.stc.jcsre.ETD** and **com.stc.jcsre.ETDConstants**.

Methods of the MQSeriesETD class

These methods are described in detail on the following pages

connectToQueueManager

Description

Creates a connection to the queue manager using the specified parameters. As a by-product it also selects as the current queue manager. This should only be called in Non-Transactional mode.

Note: *A connection to the queue manager specified in the configuration is automatically done. You need not call this method unless you want to connect to another queue manager in the collaboration.*

Syntax

```
public void connectToQueueManager(String sQueueMgrName, String Host,
        int Port, String Channel, String UserID, String Pwd)
```

Parameters

Name	Type	Description
QueueMgrName	java.lang.String	The queue manager name.
Host		The host on which the QM resides.
Port		The port to which the host system QM is listening.
Channel		The channel to use.
UserID		The user's ID - if no ID is needed, leave blank.
Pwd		The user password - if no password is needed, leave blank.

Return Values

None

Throws

com.stc.common.collabService.CollabConnException

selectQueueManager

Description

Selects from one of the connected queue managers.

Syntax

```
public void selectQueueManager(java.lang.String queueMgrName)
```

Parameters

Name	Type	Description
queueMgrName	java.lang.String	The name of the queue manager to select.

Return Values

None

Throws

com.stc.common.collabService.CollabConnException

isQueueMgrConnected

Description

Determines if the current queue manager is still connected.

Syntax

```
public boolean isQueueMgrConnected()
```

Parameters

None

Return Values

boolean
True if still connected, otherwise false.

Throws

com.stc.jcsre.EBobConnectionException

getCharacterSet

Description

Returns CCSID of the queue managers codeset for the currently selected queue manager.

Syntax

```
public int getCharacterSet()
```

Parameters

None

Return Values

integer

Throws

com.stc.common.collabService.CollabConnException

getMaximumPriority

Description

Returns the maximum message priority that can be handled by the queue manager.

Syntax

```
public int getMaximumPriority()
```

Parameters

None

Return Values

integer

Throws

com.stc.common.collabService.CollabConnException

commit

Description

Commits the operations on the currently selected queue manager. It should only be called in Non-XA mode.

Syntax

```
public void commit()
```

Parameters

None

Return Values

None

Throws

com.stc.common.collabService.CollabConnException

backout

Description

Rolls back the operations on the currently selected queue manager. It should only be called in Non-XA mode.

Syntax

```
public void backout()
```

Parameters

None

Return Values

None

Throws

com.stc.common.collabService.CollabConnException

---

queueAccessOptionsClearAll

Description

Clears all flags

Syntax

```
public queueAccessOptionsClearAll()
```

Parameters

None

Return Values

None

Throws

None

---

accessQueue

Description

Accesses a queue on the current queue manager. This routes down to the accessQueue method on the queue manager. The user can access more than one queue on the current queue manager. For each new queue accessed, add the queue to an internal collection so they can be selected by name later. See selectQueue( ). In addition, this method also sets that queue as the "current queue". This is similar to the concept of the current queue manager.

Syntax

```
public void accessQueue(java.lang.String queueName)
```

Parameters

Name	Type	Description
queueName	java.lang.String	The name of the queue to access on the current queue manager

Return Values

None

Throws

com.stc.common.collabService.CollabConnException

selectQueue

Description

Selects from one of the previously accessed queues.

Syntax

```
public void selectQueue(java.lang.String queueName)
```

Parameters

Name	Type	Description
queueName	java.lang.String	The name of the queue to select

Return Values

None

Throws

com.stc.common.collabService.CollabConnException

get

Description

Gets a message off the queue using the default options.

Syntax

```
public void get()
```

Parameters

None

Return Values

None

Throws

com.stc.common.collabService.CollabConnException

getWithOptions

Description

Gets a message off the queue using the GetMesageOptions (GMO).

Syntax

```
public void getWithOptions()
```

Parameters

None

Return Values

None

Throws

com.stc.common.collabService.CollabConnException

put

Description

Puts a message on the queue using the default options.

Syntax

```
public void put()
```

Parameters

None

Return Values

None

Throws

com.stc.common.collabService.CollabConnException

putWithOptions

Description

Puts a message on the queue using the PutMesageOptions (PMO).

Syntax

```
public void putWithOptions()
```

Parameters

None

Return Values

None

Throws

com.stc.common.collabService.CollabConnException

getCurrentDepth

Description

Gets the number of messages currently in the queue.

Syntax

```
public int getCurrentDepth()
```

Parameters

None

Return Values

integer

Throws

com.stc.common.collabService.CollabConnException

getMaximumDepth

Description

Gets the maximum number of messages that can exist on the current queue.

Syntax

```
public int getMaximumDepth()
```

Parameters

None

Return Values

integer
The number of messages.

Throws

com.stc.common.collabService.CollabConnException

getMaximumMessageLength

Description

Gets the maximum length of data that can exist in any one message on the current queue

Syntax

```
public int getMaximumMessageLength()
```

Parameters

None

Return Values

integer
The maximum size

Throws

com.stc.common.collabService.CollabConnException

newMessage

Description

Destroys and then recreate the message object. After doing a get, this must be called first, before doing another get. (See **The newMessage Method** on page 53.)

Syntax

```
public void newMessage()
```

Parameters

None

Return Values

None

Throws

None

9.2 GMO Class

GMO class methods are located under the **Queue node**.

The GMO class is defined as:

```
public class GMO
```

com.stc.eways.MQSeriesETD.GMO

Methods of the GMO class

These methods are described in detail on the following pages:

optionsClearAll on page 104   **setUnlimitedWait** on page 105

setWaitValue on page 105   **matchOptionsClearAll** on page 106

optionsClearAll

Description

Clears all option flags.

Syntax

```
public void optionsClearAll()
```

Parameters

None

Return Values

None

Throws

None

setWaitValue

Description

Specifies a specific number of milliseconds to wait.

Syntax

```
public void setWaitValue(int v)
```

Parameters

Name	Type	Description
v	integer	the number of milliseconds

Return Values

None

Throws

None

setUnlimitedWait

Description

Sets the wait time to MQWI_UNLIMITED.

Syntax

```
public void setUnlimitedWait()
```

Parameters

None

Return Values

None

Throws

None

matchOptionsClearAll

Description

Clears all match options flags set so far and set match options to MQMO_NONE.

Syntax

```
public void matchOptionsClearAll()
```

Parameters

None

Return Values

None

Throws

None

9.3   PMO Class

PMO class methods are located under the **Queue node**

The PMO class is defined as:

```
public class GMO
```

com.stc.eways.MQSeriesETD.PMO

Methods of the PMO class

These methods are described in detail on the following pages:

optionsClearAll on page 104

9.4   Message Class

Message class methods are located under the **Message node**.

The Message class is defined as:

```
public class Message
```

com.stc.eways.MQSeriesETD.Message

Methods of the Message class

These methods are described in detail on the following pages:

getTotalMessageLength on page 107   **readUInt2** on page 118

getTotalMessageLength

Description

If MQQueue.get() fails with a message-truncated error code, getTotalMessageLength( ) reports the total number of bytes in the stored message on the queue.

Syntax

```
public int getTotalMessageLength()
```

Parameters

None

Return Values

integer
The number of bytes of the message as stored on the message queue

Throws

None

getMessageLength

Description

Reports the total number of bytes in the stored message on the queue.

Syntax

```
public int getMessageLength()
```

Parameters

None

Return Values

None

Throws

com.stc.common.collabService.CollabConnException

getDataLength

Description

Reports the number of bytes of data remaining to be read in the message.

Syntax

```
public int getDataLength()
```

Parameters

None

Return Values

integer
The number, in bytes, of message data remaining to be read.

Throws

com.stc.common.collabService.CollabConnException

seek

Description

Relocates the cursor to the absolute position in the message buffer given by *pos*. Following reads and writes will act at this position in the buffer.

Syntax

```
public void seek(int pos)
```

Parameters

Name	Type	Description
pos	integer	Gives the absolute position in the message buffer.

Return Values

None

Throws

com.stc.common.collabService.CollabConnException

setDataOffset

Description

Relocates the cursor to the absolute position in the message buffer. setDataOffset ( ) is equivalent to seek(), allowing for cross-language compatibility with the other MQSeries APIs.

Syntax

```
public void setDataOffset(int offset)
```

Parameters

Name	Type	Description
offset	integer	Gives the absolute position in the message buffer.

Return Values

None

Throws

com.stc.common.collabService.CollabConnException

getDataOffset

Description

Returns the current position of the cursor within the message, that is the point at which read and write operations take effect.

Syntax

```
public int getDataOffset()
```

Parameters

None

Return Values

Integer
The current cursor position.

Throws

com.stc.common.collabService.CollabConnException

clearMessage

Description

Discards data in the message buffer and reset the data offset to zero.

Syntax

```
public void clearMessage()
```

Parameters

None

Return Values

None

Throws

com.stc.common.collabService.CollabConnException

getVersion

Description

Returns the version of the current structure.

Syntax

```
public int getVersion()
```

Parameters

None

Return Values

integer

The version of the structure in use.

Throws

None

resizeBuffer

Description

Clues the MQMessage object as to the size of buffer that may be necessary for subsequent get operations. When a message contains message data, and the new size is less than the current size, the message data is truncated.

Syntax

```
public void resizeBuffer(int size)
```

Parameters

Name	Type	Description
size	integer	The size of the buffer

Return Values

integer
The new message size.

Throws

com.stc.common.collabService.CollabConnException

readBoolean

Description

Reads a (signed) byte from the present position in the message buffer.

Syntax

```
public boolean readBoolean()
```

Parameters

None

Return Values

boolean
A byte from the current position in the message buffer

Throws

com.stc.common.collabService.CollabConnException

readChar

Description

Reads a Unicode character from the present position in the message buffer.

Syntax

```
public char readChar()
```

Parameters

None

Return Values

character
Unicode character from the current position in the message buffer.

Throws

com.stc.common.collabService.CollabConnException

readDouble

Description

Reads a double from the present position in the message buffer. Actions are determined by the value of the encoding member variable. MQC.MQENC_FLOAT_S390 reads a System/390 format floating point number. MQC.MQENC_FLOAT_IEEE_NORMAL and MQC.MQENC_FLOAT_IEEE_REVERSED read IEEE standard doubles in big-endian and little-endian formats respectively.

Syntax

```
public double readDouble()
```

Parameters

None

Return Values

double

Throws

com.stc.common.collabService.CollabConnException

readFloat

Description

Reads a float from the present position in the message buffer. Actions are determined by the value of the encoding member variable. MQC.MQENC_FLOAT_S390 reads a System/390 format floating point number. MQC.MQENC_FLOAT_IEEE_NORMAL and MQC.MQENC_FLOAT_IEEE_REVERSED read IEEE standard floats in big-endian and little-endian formats respectively.

Syntax

```
public float readFloat()
```

Parameters

None

Return Values

float

Throws

com.stc.common.collabService.CollabConnException

readFully

Description

Fills the byte array b with data from the message buffer.

Fill *len* elements of the byte array b with data from the message buffer, starting at offset *off*.

Syntax

```
public void readFully(byte b[])
public void readFully(byte b[], int off, int len)
```

Parameters

Name	Type	Description
b []	byte	Fill the byte array b with data from the message buffer.

Name	Type	Description
b []	byte	Fill the byte array b with data from the message buffer.
off	integer	Start fill at offset *off*.
len	integer	Fill *len* elements of the byte array b with data from the message buffer.

Return Values

None

Throws

com.stc.common.collabService.CollabConnException

readInt

Description

Reads an integer from the present position in the message buffer. Actions are determined by the value of the encoding member variable. A value of MQC.MQENC_INTEGER_NORMAL reads a big-endian integer, a value of MQC.MQENC_INTEGER_REVERSED reads a little-endian integer.

Syntax

```
public int readInt()
```

Parameters

None

Return Values

integer
An integer from the current position in the message buffer

Throws

com.stc.common.collabService.CollabConnException

readInt4

Description

Equivalent to readInt(), readInt4 is provided for cross-language MQSeries API compatibility.

Syntax

public int **readInt4**()

Parameters

None

Return Values

integer
An integer from the current position in the message buffer

Throws

com.stc.common.collabService.CollabConnException

readLine

Description

Converts from the codeset defined in the characterSet member variable to Unicode, then reads in a line that has been terminated by \n, \r, \r\n, or EOF.

Syntax

```
public String readLine()
```

Parameters

None

Return Values

String

Throws

com.stc.common.collabService.CollabConnException

readLong

Description

Reads a long from the present position in the message buffer. Actions are determined by the value of the encoding member variable. A value of MQC.MQENC_INTEGER_NORMAL reads a big-endian long, a value of MQC.MQENC_INTEGER_REVERSED reads a little-endian long.

Syntax

```
public long readLong()
```

Parameters

None

Return Values

long

Throws

com.stc.common.collabService.CollabConnException

readInt8

Description

Equivalent to readlong(), readInt8 is provided for cross-language MQSeries API compatibility.

Syntax

```
public int readInt8()
```

Parameters

None

Return Values

>   **long**

Throws

>   **com.stc.common.collabService.CollabConnException**

---

readObject

Description

>   Reads an object, its class, class signature, and the value of the non-transient and non-static fields of the class.

Syntax

```
public Object readObject()
```

Parameters

>   None

Return Values

>   **Object**

Throws

>   **com.stc.common.collabService.CollabConnException**

---

readShort

Description

>   Reads a short from the present position in the message buffer. Actions are determined by the value of the encoding member variable. A value of MQC.MQENC_INTEGER_NORMAL reads a big-endian short, a value of MQC.MQENC_INTEGER_REVERSED reads a little-endian short.

Syntax

```
public short readShort()
```

Parameters

>   None

Return Values

>   **Short**

Throws

>   **com.stc.common.collabService.CollabConnException**

readInt2

Description

Equivalent to readshort(), readInt2 is provided for cross-language MQSeries API compatibility.

Syntax

```
public short readInt2()
```

Parameters

None

Return Values

short

Throws

com.stc.common.collabService.CollabConnException

readUTF

Description

Reads a UTF string, prefixed by a 2-byte length field, from the present position in the message buffer.

Syntax

```
public String readUTF()
```

Parameters

None

Return Values

String

Throws

com.stc.common.collabService.CollabConnException

readUnsignedByte

Description

Reads an unsigned byte from the present position in the message buffer.

Syntax

```
public int readUnsignedByte()
```

Parameters

None

Return Values

integer

Throws

com.stc.common.collabService.CollabConnException

readUnsignedShort

Description

Reads an unsigned short from the present position in the message buffer. Actions are determined by the value of the encoding member variable. A value of MQC.MQENC_INTEGER_NORMAL reads a big-endian unsigned short, a value of MQC.MQENC_INTEGER_REVERSED reads a little-endian unsigned short.

Syntax

```
public int readUnsignedShort()
```

Parameters

None

Return Values

integer

Throws

com.stc.common.collabService.CollabConnException

readUInt2

Description

Equivalent to readUnsignedShort(), readUInt2 is provided for cross-language MQSeries API compatibility.

Syntax

```
public int readUInt2()
```

Parameters

None

Return Values

integer

Throws

com.stc.common.collabService.CollabConnException

readString

Description

Reads a string in the codeset defined by the characterSet member variable. Convert the string into Unicode.

Syntax

```
public String readString(int length)
```

Parameters

Name	Type	Description
length	integer	The number of characters to read (not the same as the number of bytes)

Return Values

String

Throws

com.stc.common.collabService.CollabConnException

readDecimal2

Description

Reads a 2-byte packed decimal number (-999 to 999). Actions are determined by the value of the encoding member variable. A value of MQC.MQENC_DECIMAL_NORMAL reads a big-endian packed decimal number, and a value of MQC.MQENC_DECIMAL_REVERSED reads a little-endian packed decimal number.

Syntax

```
public short readDecimal2()
```

Parameters

None

Return Values

short

Throws

com.stc.common.collabService.CollabConnException

readDecimal4

Description

Reads a 4-byte packed decimal number (-9999999 to 9999999). Actions are determined by the value of the encoding member variable. A value of MQC.MQENC_DECIMAL_NORMAL reads a big-endian packed decimal number, and a value of MQC.MQENC_DECIMAL_REVERSED reads a little-endian packed decimal number.

Syntax

```
public int readDecimal4()
```

Parameters

None

Return Values

integer

Throws

com.stc.common.collabService.CollabConnException

readDecimal8

Description

Reads a 8-byte packed decimal number (-999999999999999 to 999999999999999). Actions are determined by the value of the encoding member variable. A value of MQC.MQENC_DECIMAL_NORMAL reads a big-endian packed decimal number, and a value of MQC.MQENC_DECIMAL_REVERSED reads a little-endian packed decimal number.

Syntax

```
public long readDecimal8()
```

Parameters

None

Return Values

long

Throws

com.stc.common.collabService.CollabConnException

setVersion

Description

Sets the version of the structure to be used. Values may include MQC.MQMD_VERSION_1 or MQC.MQMD_VERSION_2. This method is used when it

is necessary to force a client to use a version 1 structure when connected to a queue manager that is able to handling version 2 structures. In all other situations, the client determines the correct version by querying the queue manager's capabilities.

Syntax

```
public void setVersion(int version)
```

Parameters

Name	Type	Description
version	integer	The version number

Return Values

> **String**

Throws

> None

skipBytes

Description

Advances n bytes in the message buffer. Block until all the bytes are skipped, the end of message buffer is detected, or an exception is thrown.

Syntax

```
public int skipBytes(int n)
```

Parameters

Name	Type	Description
n	integer	Move forward n bytes in the message buffer.

Return Values

> **integer**
> Returns the number of bytes skipped, which is always n.

Throws

> **com.stc.common.collabService.CollabConnException**

write

Description

Writes a byte, an array of bytes, or a series of bytes into the message buffer at the present position. *len* bytes will be written, taken from offset *off* in the array b.

Syntax

```
public void write(int b)
public void write(byte b[])
public void write(byte b[], int off, int len)
```

Parameters

Name	Type	Description
b	integer	The number of characters to read (not the same as the number of bytes)

Name	Type	Description
b[]	byte	The number of characters to read (not the same as the number of bytes)

Name	Type	Description
b[]	byte	The number of characters to read (not the same as the number of bytes)
off	integer	The offset in the array
len	integer	The number of bytes to be written

Return Values

None

Throws

com.stc.common.collabService.CollabConnException

writeBoolean

Description

Writes a boolean into the message buffer at the present position.

Syntax

```
public void writeBoolean(boolean v)
```

Parameters

Name	Type	Description
v	boolean	The boolean value

Return Values

None

Throws

com.stc.common.collabService.CollabConnException

writeByte

Description

Writes a byte into the message buffer at the present position.

Syntax

```
public void writeByte(int v)
```

Parameters

Name	Type	Description
v	integer	The number of characters to read (not the same as the number of bytes)

Return Values

None

Throws

com.stc.common.collabService.CollabConnException

writeBytes

Description

Writes the string to the message buffer as a sequence of bytes. Each character is written out in sequence by discarding its high eight bits.

Syntax

```
public void writeBytes(String s)
```

Parameters

Name	Type	Description
s	String	The string of bytes.

Return Values

None

Throws

com.stc.common.collabService.CollabConnException

writeChar

Description

Writes a Unicode character into the message buffer at the present position.

Syntax

```
public void writeChar(int v)
```

Parameters

Name	Type	Description
v	integer	The unicode value.

Return Values

None

Throws

com.stc.common.collabService.CollabConnException

writeChars

Description

Writes a string as a sequence of Unicode characters into the message buffer at the current position.

Syntax

```
public void writeChars(String s)
```

Parameters

Name	Type	Description
s	String	The string value.

Return Values

None

Throws

com.stc.common.collabService.CollabConnException

writeDouble

Description

Writes a double into the message buffer at the present position. The actions of this method are determined by the value of the encoding member variable.

A Value of MQC.MQENC_FLOAT_IEEE_NORMAL or
MQC.MQENC_FLOAT_IEEE_REVERSED write IEEE standard floats in big-endian and
little-endian formats respectively.

A value of MQC.MQENC_FLOAT_S390 writes a System/390 format floating point
number. The range of IEEE doubles is greater than the range of S/390 double precision
floating point numbers. Very large numbers cannot be converted.

Syntax

```
public void writeDouble(double v)
```

Parameters

Name	Type	Description
v	double	The number of characters to read (not the same as the number of bytes)

Return Values

None

Throws

com.stc.common.collabService.CollabConnException

writeFloat

Description

Writes a float into the message buffer at the present position. The actions of this method
are determined by the value of the encoding member variable.

A Value of MQC.MQENC_FLOAT_IEEE_NORMAL or
MQC.MQENC_FLOAT_IEEE_REVERSED write IEEE standard floats in big-endian and
little-endian formats respectively.

A value of MQC.MQENC_FLOAT_S390 will write a System/390 format floating point
number.

Syntax

```
public void writeFloat(float v)
```

Parameters

Name	Type	Description
length	integer	The float value

Return Values

None

Throws

com.stc.common.collabService.CollabConnException

writeInt

Description

Writes an integer into the message buffer at the present position. The actions of this method are determined by the value of the encoding member variable.

A value of MQC.MQENC_INTEGER_NORMAL writes a big-endian integer. A value of MQC.MQENC_INTEGER_REVERSED writes a little-endian integer.

Syntax

```
public void writeInt(int v)
```

Parameters

Name	Type	Description
v	integer	The float value

Return Values

None

Throws

com.stc.common.collabService.CollabConnException

writeLong

Description

Writes a long into the message buffer at the present position. The actions of this method are determined by the value of the encoding member variable.

A value of MQC.MQENC_INTEGER_NORMAL writes a big-endian long. A value of MQC.MQENC_INTEGER_REVERSED writes a little-endian long.

Syntax

```
public void writeLong(long v)
```

Parameters

Name	Type	Description
v	long	The long value

Return Values

None

Throws

com.stc.common.collabService.CollabConnException

writeObject

Description

Writes the specified object, object class, class signature, and the values of the non-transient and non-static fields of the class and all its supertypes.

Syntax

```
public void writeObject(Object obj)
```

Parameters

Name	Type	Description
obj	object	The object value

Return Values

None

Throws

com.stc.common.collabService.CollabConnException

writeShort

Description

Writes a short into the message buffer at the present position. The actions of this method are determined by the value of the encoding member variable.

A value of MQC.MQENC_INTEGER_NORMAL writes a big-endian short. A value of MQC.MQENC_INTEGER_REVERSED writes a little-endian short.

Syntax

```
public void writeShort(int v)
```

Parameters

Name	Type	Description
v	integer	The integer value

Return Values

None

Throws

com.stc.common.collabService.CollabConnException

writeDecimal2

Description

Writes a 2-byte packed decimal format number into the message buffer at the present position. The actions of this method are determined by the value of the encoding member variable.

A value of MQC.MQENC_DECIMAL_NORMAL writes a big-endian packed decimal. A value of MQC.MQENC_DECIMAL_REVERSED writes a little-endian packed decimal.

Syntax

```
public void writeDecimal2(short v)
```

Parameters

Name	Type	Description
v	short	The 2-byte decimal value.

Return Values

None

Throws

com.stc.common.collabService.CollabConnException

writeDecimal4

Description

Writes a 4-byte packed decimal format number into the message buffer at the present position. The actions of this method are determined by the value of the encoding member variable.

A value of MQC.MQENC_DECIMAL_NORMAL writes a big-endian packed decimal. A value of MQC.MQENC_DECIMAL_REVERSED writes a little-endian packed decimal.

Syntax

```
public void writeDecimal4(int v)
```

Parameters

Name	Type	Description
v	integer	The 4-byte decimal value.

Return Values

None

Throws

com.stc.common.collabService.CollabConnException

writeDecimal8

Description

Writes an 8-byte packed decimal format number into the message buffer at the present position. The actions of this method are determined by the value of the encoding member variable.

A value of MQC.MQENC_DECIMAL_NORMAL writes a big-endian packed decimal. A value of MQC.MQENC_DECIMAL_REVERSED writes a little-endian packed decimal.

Syntax

```
public void writeDecimal8(long v)
```

Parameters

Name	Type	Description
v	long	The 8-byte decimal value.

Return Values

None

Throws

com.stc.common.collabService.CollabConnException

writeUTF

Description

Writes a UTF string, prefixed by a 2-byte length field, into the message buffer at the present position.

Syntax

```
public void writeUTF(String str)
```

Parameters

Name	Type	Description
str	String	The string value

Return Values

None

Throws

com.stc.common.collabService.CollabConnException

writeString

Description

Writes a string into the message buffer at the present position, converting it to the codeset identified by the characterSet member variable.

Syntax

```
public void writeString(String str)
```

Parameters

Name	Type	Description
str	String	The string value

Return Values

None

Throws

com.stc.common.collabService.CollabConnException

Appendix A (JMS)

0.1 Mapping Between JMS Standard Header Items and MQSeries Header Fields

JMS Standard header items and their equivalent MQSeries header fields can be set using the Collaboration Rules Editor. For information on mapping between JMS header items and MQSeries header fields see IBM MQSeries online documentation at: **http://www-4.ibm.com/software/ts/mqseries/library/manual01/csqzaw07/csqzaw07tfrm.htm**

Table 20, at the above Web site shows how JMS header fields are used to set or get MQSeries header fields (only some of which are available using this procedure). The Collaboration Rules Editor sets the header properties by calling **readProperty()** or **writeProperty()**. To map these properties do the following:

1 In the Collaboration Rules Editor, click **rule** on the Business Rules toolbar to create a rule at the appropriate place. Select the new rule.

2 Form the Source Events or Destination Events panes, drag-and-drop **readProperty** into the Rule Properties, Rules window(seeFigure 1).

Figure 1 Collaboration Rules Editor
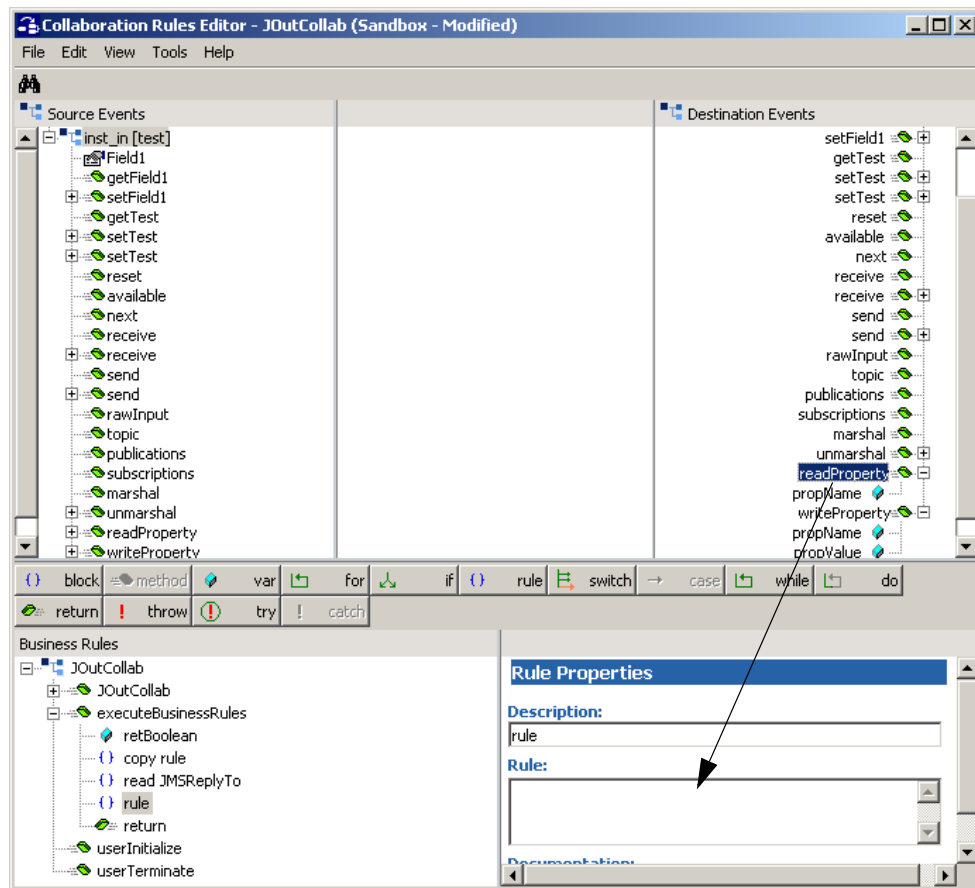
3 A dialog box opens prompting for the **property name**. Enter the property as a string with quotes. For example, **"JMSDeliveryMode"**. The following code displays in the Rule Properties, Rule window:

```
getinst_out().readProperty("JMSDeliveryMode")
```

4 Click rule on the toolbar. A new rule expression appears in the Business Rules pane.

5 Form the Source Events or Destination Events panes, drag-and-drop **writeProperty** into the Rule Properties, Rules window. A dialog box opens prompting for the **property name** and the **property value** (see Figure 2).

Figure 2   Parameters for writeProperty()
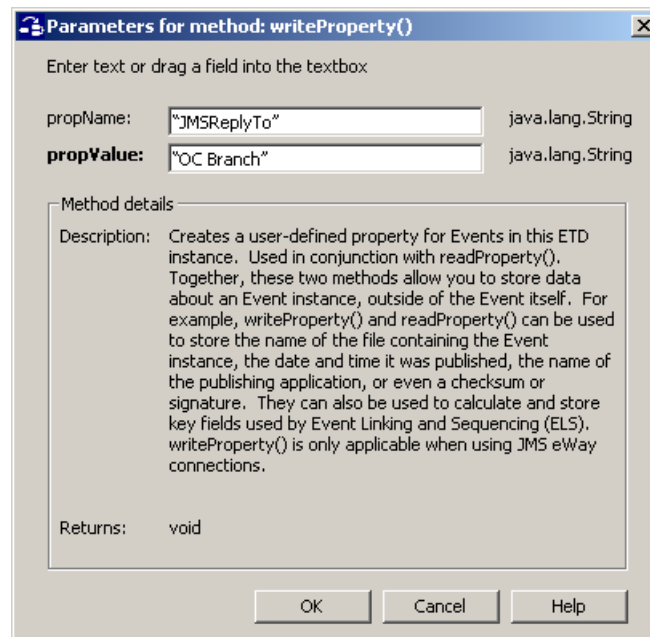
6   Enter the property as a string with quotes. For example, **"JMSReplyTo"** as the
    property name, and **"OC Branch"** as the **property value**. The following code
    appears in the Rule window:

```
getinst_out().writeProperty("JMSReplyTo","OC Branch")
```

7   A description stating the purpose of this rule can be added in the Rule Properties,
    Description field and will display in the Business Rules pane

Note: *For detailed information on creating Collaboration Rules using the Java*
          *Collaboration Rules Editor see the **e*Gate Integrator User's Guide**.*

Index

Q

queue access **51**
Queue Manager Name **61**
Queue Manager Name (JMS) **25**

R

readData **55**
readFully **55**
reason **93**
reason code **49**
Remote Debugging Port Number **20**
Report JVM Info and all Class Loads **20**

S

sample schema **48**, **93**
    OS/390 **95**
setting CLASSPATH variables **47**, **92**
    for Windows **47**, **92**
Suspend Option for Debugging **21**
system requirements
    external system requirements **10**
        patches **10**

T

Transaction Type **60**
transaction type
    Non-Transactional **60**
    XA-compliant. **60**
Transaction Type (JMS) **23**
Transport Type (JMS) **25**

U

UNIX
    installation **14**

W

wait interval **63**
wait timeout **63**
Windows NT **12**

X

XA **60**
XA-compliant transaction type **60**