SeeBeyond ICAN Suite

e*Way Intelligent Adapter for WebSphere Application Server User's Guide

Release 5.0.5 for Schema Run-time Environment (SRE)

Java Version

Contents

Chapter 3

Configuration 30

Chapter 4

Implementation 52

Chapter 1

Introduction

This chapter provides a brief introduction to the SeeBeyond e*Way Intelligent Adapter for WebSphere Application Server. It includes:

- A general overview of the e*Way's functionality.
- A general overview of the WebSphere® Application Server.
- Information about supported operating systems and system requirements.

1.1 Intended Reader

The reader of this guide is presumed:

- To be a developer or system administrator with the responsibility of maintaining e*Gate.
- To have high-level knowledge of Windows or UNIX operations and administration.
- To be familiar with WebSphere Application Server administration.
- To have high-level knowledge of Java™, J2EE™ Services, JMS™, and Enterprise JavaBeans™.

1.2 Overview

The e*Way Intelligent Adapter for WebSphere Application Server

The e*Way Intelligent Adapter for WebSphere Application Server (WebSphere Application Server e*Way) provides a communication interface that facilitates integration of external applications and systems with IBM's WebSphere Application Server via e*Gate Integrator.

The WebSphere Application Server e*Way supports asynchronous enterprise messaging from e*Gate to WebSphere and from WebSphere to e*Gate.

WebSphere Application Server

IBM's WebSphere Application Server is used to build new applications with graphical interfaces. It provides an architecture for building business logic in re-usable components so that a Web server can easily access data.

The WebSphere Application Server is J2EE (Java 2, Enterprise Edition) 1.3 compliant and uses J2EE services. WebSphere's J2EE compliant architecture enables you to create applications that provide standardization, scalability, security, and reusability.

The WebSphere Application Server uses Enterprise JavaBeans (EJBs). EJBs are the units of work that an Application Server is responsible for and exposes to the external world.

The WebSphere Application Server allows you to build EJBs and deploy them, making them available to other applications on various systems. These EJBs are Java programs written by the developer and deployed to the WebSphere Application Server. The WebSphere Application Server offers services such as connectivity, business logic, re-usability, security, concurrency (access is serialized), and transactionality (uses XA to assure a successful message transfer or update or message rollback).

Chapter 2, discusses J2EE services, building and deploying EJBs, and enterprise messaging. **Chapter 2** also discusses how the WebSphere Application Server e*Way supports asynchronous enterprise messaging between e*Gate and WebSphere.

1.3 Supported Operating Systems

The WebSphere Application Server e*Way is available on the following operating systems:

- Windows 2000, Windows XP, and Windows Server 2003
- IBM AIX 5.1L and 5.2
- Sun Solaris 8 and 9

1.4 System Requirements

To use the WebSphere Application Server e*Way, you need the following:

- An e*Gate Participating Host.
- 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.

Note: *Open and review the* ***Readme.txt*** *for the WebSphere Application Server e*Way for any additional requirements prior to installation. The* ***Readme.txt*** *is located in* ***..\samples\ewwebsphere\java-WAS5.1***

1.4.1. External System Requirements

- IBM WebSphere® Application Server 5.0

Chapter 2

Services, XA Transactions, and Enterprise Messaging

This chapter describes the J2EE services and enterprise messaging services that are used by the WebSphere Application Server and the WebSphere Application Server e*Way. It includes:

- A general overview of J2EE services.
- A general overview of enterprise messaging services.
- A general overview of how the e*Way works with the WebSphere Application Server via asynchronous messaging.
- Information about asynchronous processing.
- Information about how the e*Way handles messaging using SeeBeyond JMS.
- A description of how the e*Way works with the WebSphere Application Server.

2.1 J2EE Services

The following sections provide an overview to the JNDI™, JMS and EJB services.

2.1.1. Java Naming and Directory Interface (JNDI)

Java Naming and Directory Interface™ (JNDI) is a set of APIs that allows a Java program to store objects and lookup objects using multiple naming services in a standard manner. A naming service may be LDAP, a file system, or an RMI registry. Each naming service has a corresponding provider implementation that can be used with JNDI. JNDI can span across naming services in a federated naming service. This allows any Java code using JNDI to be portable against any naming service. For example, no code changes should be needed by the Java client code to run against an LDAP server or an RMI registry.

WebSphere Naming Service

To see all the registered names in the WebSphere Naming Service, you can run the **dumpNameSpace.bat** script or open it up in a text editor. For more information on the

WebSphere Naming Service, see the *IBM WebSphere Application Server Version 5.0 Handbook*.

2.1.2. Java Messaging Service (JMS)

The Java Messaging Service (JMS) allows client portability with any JMS implementation. The clients do not communicate with each other directly. Instead, the clients send messages to each other via JMS. Each client in a JMS environment connects to a messaging server. The messaging server facilitates the flow of messages among all clients and guarantees that all messages arrive at the appropriate destinations.

There are two possible destinations that a client sends messages to or receives messages from. They are **Topic** and **Queue** (see Figure 1 and Figure 2). The difference between a Topic and a Queue is that all subscribers to a Topic receive the same message when the message is published while only one subscriber to a Queue receives a message when the message is sent. For more information see **SeeBeyond JMS** on page 19.

Figure 1 Topic - The Publish-Subscribe Model

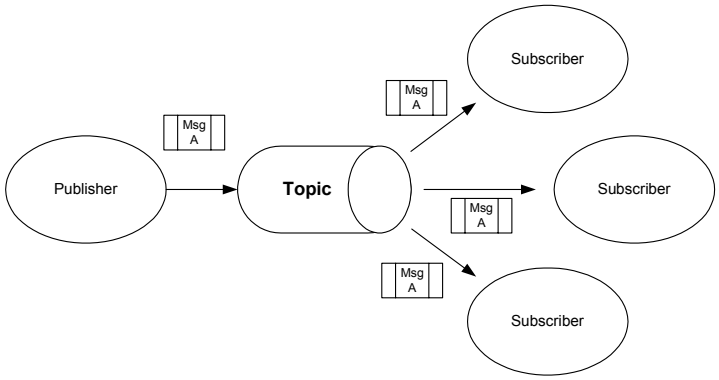
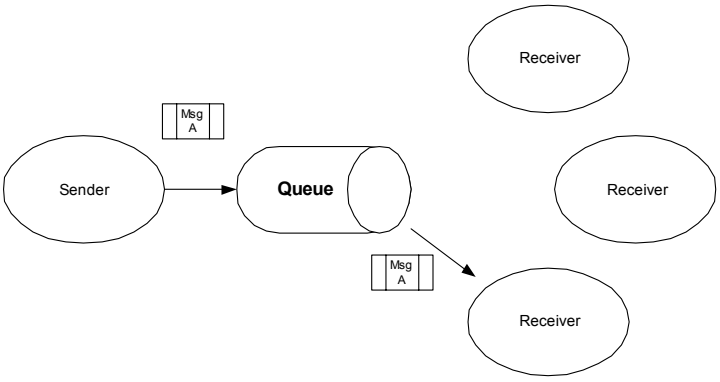


Figure 1 shows multiple subscribers receiving the same messages when the publisher publishes the message to a Topic. This is the Publish-Subscribe model.

Figure 2 Queue - The Point-to-Point Model



In contrast, the Point-to-Point model (Figure 2) allows for only one of the receivers to receives the message when a sender sends a message to a Queue.

2.1.3. Enterprise JavaBeans (EJBs)

Enterprise JavaBean Architecture

Enterprise JavaBean architecture supports the development and deployment of component-based distributed business applications. Applications written using the Enterprise JavaBean architecture are scalable, transactional, and multi-user secure. These applications may be written once, and then deployed on any server platform that supports the *Enterprise Java Beans 2.1 Specification*.

If a developer writes an EJB that adheres to the Enterprise JavaBean specification, the EJB can be deployed on any EJB container regardless of the type of software container or application server. The EJB developer does not need to write any code relating to transactions or threads. These services are provided by the container in which an EJB is deployed. The EJB developer must define the attributes of the EJB in its deployment descriptor in order to take advantage of these services offered by the container.

Message Driven Beans

A Message Driven Bean (MDB) is a type of EJB that handles asynchronous subscription and publication of JMS messages. An MDB is often compared to a Stateless Session Bean because it does not have any state context. An MDB differs from Session and Entity Beans in that it has no local/remote or localhome/home interfaces.

A MDB is a specialized EJB that triggers whenever there is activity on a specific queue. A MDB is not exposed to a client. Rather, it simply subscribes to a Topic or a Queue, receives messages from the container via the Topic or Queue, and then processes the messages it receives from the container. MDBs enable client components to send messages to an EJB container without having to wait for a reply.

An MDB implementation requires two interfaces: **javax.ejb.MessageBean** and **javax.jms.MessageListener**. At a minimum, the MDB must implement the **setMessageDrivenContext**, **ejbCreate**, and **ejbRemove** methods from the **javax.ejb.MessageBean** interface. In addition, the MDB must implement the **onMessage** method of the **javax.jms.MessageListener** interface. The container calls the **onMessage** method, passing in a **javax.jms.Message** when a message is available for the MDB.

Session Beans

A Session Bean is another type of EJB. The Session Bean consists of the remote, home, and bean classes. A client gets a reference to the Session Bean's home interface in order to create the Session Bean remote object, which is essentially the bean's factory. The Session Bean is exposed to the client with the remote interface. The client uses the remote interface to invoke the bean's methods. The actual implementation of the Session Bean is done with the bean class.

Entity Beans

An Entity Bean, like a Session Bean, consists of the remote, home, and bean classes. The client references the Entity Bean's home interface in order to create the Entity Bean remote object (essentially the bean's factory). The Entity Bean is exposed to the client with the remote interface which the client uses to invoke the bean's methods. The implementation of the Entity Bean is done with the bean class.

2.2 Overview of e*Gate and WebSphere Application Server Messaging

e*Gate interacts with the WebSphere Application Server via asynchronous messaging. Asynchronous messaging interaction means that a request is sent but the sender does not wait for a response. It can be thought of as analogous to a mail message in which mail is sent and forgotten until sometime later when a response is received.

Figure 3 shows how the WebSphere Application Server e*Way supports asynchronous enterprise messaging from e*Gate to WebSphere.

Figure 3 Message flow from e*Gate to WebSphere

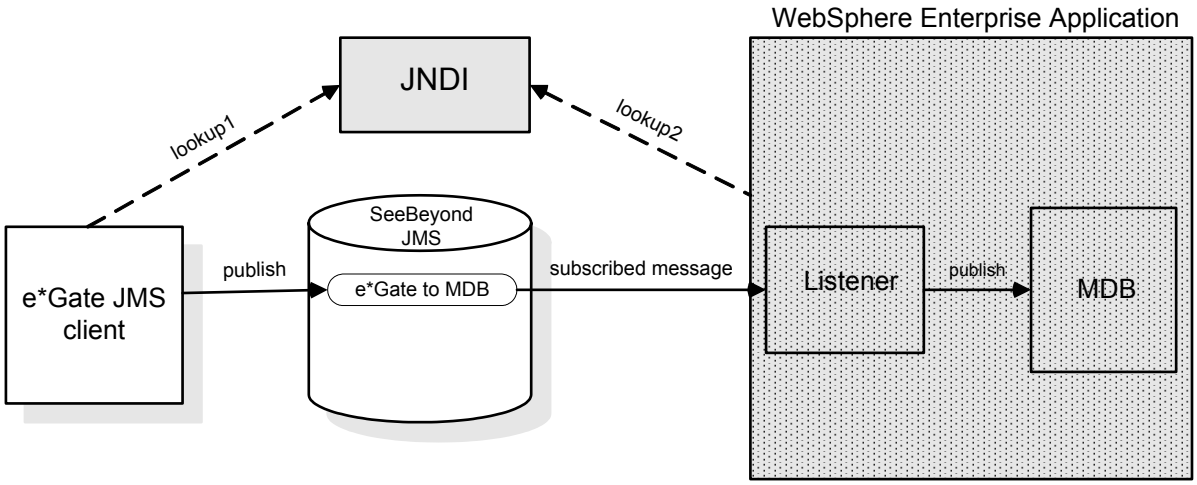
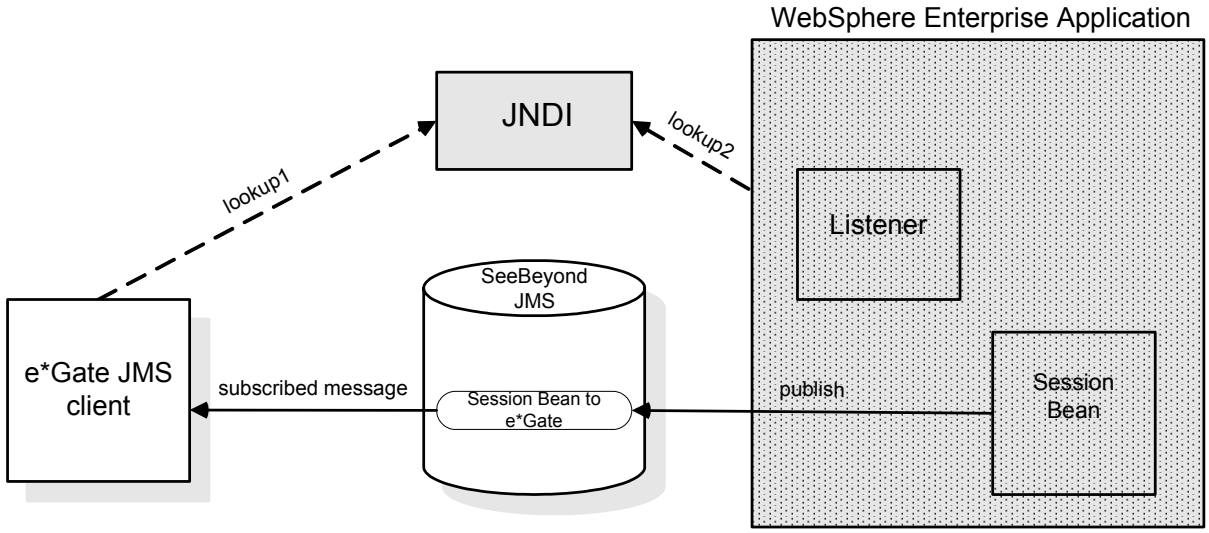


Figure 4 shows how the WebSphere Application Server e*Way supports asynchronous enterprise messaging from WebSphere to e*Gate.

Figure 4 Message flow from WebSphere to e*Gate



2.3 XA Transactions

XA transactions require XA-compliant software systems. If cooperating software systems are XA-compliant, they guarantee that for each unit of data transferred between systems:

- No data is lost.
- No unit of data is duplicated.

2.3.1. XA Transaction Process Overview

The X/Open XA Specification defines the interactions between the Transaction Manager (TM) and the Resource Manager (RM). The TM, also known as the XA Coordinator, manages the XA or global transactions. The RM manages a particular resource such as a database or a JMS system. In addition, an XA Resource exposes a set of methods or functions for managing the resource.

In order to be involved in an XA transaction, an XA Resource must make itself known to the TM. This process is called enlistment. Once an XA Resource is enlisted, the TM ensures that the XA Resource takes part in a transaction and makes the appropriate method calls on the XA Resource during the lifetime of the transaction.

For an XA transaction to complete, all the RMs participate in a two-phase commit. A commit in an XA transaction is called a two-phase commit because there are two passes made in the committing process. In the first pass, the TM asks each of the RMs whether they will encounter any problems committing the transaction. If any RM objects to committing the transaction, then all work done by any party on any resource involved in the XA transaction must all be rolled back. The TM calls the rollback method on each of the enlisted XA Resources. However, if no RMs object to committing, then the second

pass involves the TM actually calling commit on each of the enlisted XA Resources. This process guarantees a transaction that can span multiple resources.

Guaranteed Exactly Once Delivery

Occasionally, a failure condition can occur in data transfer systems. Data can be lost through system errors. To compound these problems, identical data is often delivered more than once after system restarts.

The WebSphere Application Server e*Way allows you to guarantee that each unit of data is delivered exactly once. In e*Gate, this feature is called Guaranteed Exactly Once Delivery (GEOD). Along with e*Gate, the e*Way guarantees exact delivery by utilizing the XA protocol.

2.3.2. XA Transactions With SeeBeyond JMS and the WebSphere Application Server

Both SeeBeyond JMS and WebSphere Application Server implement the X/Open XA interface specifications. Because both systems support XA, the EJBs running inside the WebSphere container can either send or receive messages via either a topic or a queue. When running in XA, these EJBs can also participate in global transactions involving other EJBs.

In a Container Managed Transaction (CMT), the WebSphere container interacts closely with the TM. This interaction is so close that the transactions can be transparent to an EJB developer. Since the transactional attributes of EJBs are defined through their deployment descriptors, they allow the container to transparently handle the XA transactions on behalf of the EJBs.

The WebSphere TM coordinates the XA transactions. The SeeBeyond JMS XA Resource is enlisted to the XA transaction, making the WebSphere TM aware that the SeeBeyond JMS XA Resource is involved in the XA transaction.

You must carefully and intentionally design MDBs and Session Beans in order to support XA transactions.

Designing an MDB to Support XA Transactions

MDBs handle messages read from JMS destinations within the scope of a transaction. If transaction handling is specified for a JMS destination, the JMS listener starts a global transaction before it reads any incoming message from that destination. When the MDB processing has finished, the JMS listener commits or rolls back the transaction using Java Transaction API (JTA) transaction control.

When designing an MDB with a CMT, it must be carefully designed to support XA transactions. The MDB design must meet the following requirements:

- The MDB's connection factory must be an XA connection factory.
- The MDB's listening port must have the JNDI name pointing to an XA connection factory.

Note: *All messages retrieved from a specific destination have the same transactional behavior. If non-WebSphere XA connection factories are used, then the JMS resources are not recovered if the server fails.*

Designing a Session Bean to Support XA Transactions

XA resources can be used to publish messages to SeeBeyond JMS by using standard JMS APIs in a Session Bean. The EJB server enlists the JMS session of the JMS connection to the SeeBeyond JMS server as part of a transaction. When the transaction commits, the EJB server and SeeBeyond JMS server perform a two-phase commit.

When designing a Session Bean with a CMT, it must be carefully designed to support XA transactions. The Session Bean design must meet the following requirements:

- The Session Bean's transaction type must be properly set.
- The Session Bean's resource reference must specify the logical name for the resource, specify the JNDI name, and specify the JNDI name type.
- The transaction attributes of the Session Bean method must be specified either by the bean provider or by the application assembler.

Designing a Session Bean to support XA transactions has more complex requirements than designing MDBs. The following section describes the requirements in detail, explains how to meet the design requirements, and discusses caveats to be aware of during the design process.

Configuring the Session Bean's Transaction Type

In WebSphere Application Server Developer Studio, configure the Session Bean's transaction type to **Container**.

Once properly configured, the WebSphere Application Server Developer studio puts **<transaction-type>Container</transaction-type>** in the bean's deployment descriptor.

Specifying the Logical Name, the JNDI Name, and the JNDI Resource Type

Edit the bean's Resource References section by typing a logical name. The Resource Reference uses a logical name to locate a connection factory object. These objects define connections to external resources such as SeeBeyond JMS connection factories.

1. Type a logical name for the resource. The sample schemas (see **Chapter 4**) use the logical name **SBYN/Queue/sbynConnectionFactoryXA** for the resource. In the Session Bean Java code, it enables a JNDI look up using the following:

   java:comp/env/ SBYN/Queue/sbynConnectionFactoryXA

2. In the **JNDI Name** field, type the JNDI name for the **STCXAQueueConnectionFactory**.

Note: *You must also configure the JNDI name in the WebSphere Application Server administrative console. This information is provided in* **"Configuring JMS Connection Factories for SeeBeyond JMS" on page 46***.*

3. Type **javax.jms.QueueConnectionFactory** as the resource type.

Once configured, the WebSphere Application Server Developer studio places the following in the bean's deployment descriptor:

```
<resource-ref id="ResourceRef_1042659891564">
               <res-ref-name>SBYN/Queue/sbynConnectionFactoryXA</res-ref-name>
               <res-type>javax.jms.QueueConnectionFactory</res-type>
               <res-auth>Application</res-auth>
               <res-sharing-scope>Unshareable</res-sharing-scope>
            </resource-ref>
```

The sample Session Bean Java code using the configured resources is as follows:

```
initCtx = new InitialContext();
QueueConnectionFactory  qcf = (QueueConnectionFactory) initCtx.lookup("java:comp/env/
SBYN/Queue/sbynConnectionFactoryXA);
Queue queue = (Queue) initCtx.lookup("java:comp/env/JMSQueue/sbynqueuefromWSXA");
QueueConnection qc = qcf.createQueueConnection();
qc.start();
QueueSession qs = qc.createQueueSession(true,Session.AUTO_ACKNOWLEDGE);
QueueSender sender = qs.createSender(queue);
```

Because the container manages the transactional enlistment of JMS sessions on behalf of a bean, the parameters of the **createSession** (Boolean transacted, int acknowledgeMode), **createQueueSession** (Boolean transacted, int acknowledgeMode), and **createTopicSession** (Boolean transacted, int acknowledgeMode) methods are ignored.

The *Enterprise Java Beans 2.1 Specification* recommends that the bean provider specify that a session is transacted, but provide 0 for the value of acknowledgment mode.

Bean Design and Deployment Caveats

Though the JNDI is bound to a **XAQueueConnectionFactory**, you must set **QueueConnectionFactory** as the resource type. In addition, **QueueConnectionFactory** must be used through out your Session Bean code--not **XAQueueConnectionFactory**.

Important: *You must use **QueueConnectionFactory** in your Session Bean Java code. Using **XAQueueConnectionFactory** will cause a Java class casting error.*

The JNDI name must bind to **XAQueueConnectionFactory**. If the JNDI name is not bound to an XA resource, you will get a run time exception error that reads that the "resource only implements one phase protocol but requires two phase protocol".

The reason you must select **QueueConnectionFactory** as the resource type is that the real physical connection is managed by the container. When the bean provider does a lookup, the bean provider only receives a logical reference to the physical connection. The *Enterprise Java Beans 2.1 Specification* requires the bean provider to use **javax.jms.QueueConnection** or **javax.jms.TopicConnectionFactory** interfaces as a logical reference type.

Configuring the Transaction Attributes of the Session Bean Method

The transaction attributes must be specified either by the person programming the beans or by the person deploying the beans.

If you are programming beans, use WebSphere Application Server Developer Studio to specify the attributes. If you are deploying beans, use WebSphere Application Server Application Assembly Tool. For more information, see the WebSphere Application Server Developer Studio and WebSphere Application Server Application Assembly Tool user documentation.

Once you configure the transaction attributes, the **ejb-jar.xml** file is generated with the following:

```
<assembly-descriptor id="AssemblyDescriptor_1039145481360">
         <container-transaction id="MethodTransaction_1042659891574">
            <description>XATopicSendToSbynJMS:+:</description>
            <method>
            <ejb-name>PubMsgToSbynJMSXA</ejb-name>
            <method-intf>Remote</method-intf>
            <method-name>sendToSbynJms</method-name>
            <method-params>
            <method-param>java.lang.String</method-param>
            <method-param>int</method-param>
            </method-params>
            </method>
            <trans-attribute>Required</trans-attribute>
         </container-transaction>
         <container-transaction id="MethodTransaction_1042659891575">
            <description>XAQueueSendToSBynJMS:+:</description>
            <method>
            <ejb-name>QSendMsgToSBynJMSXA</ejb-name>
            <method-intf>Remote</method-intf>
            <method-name>sendToSbynJms</method-name>
            <method-params>
            <method-param>java.lang.String</method-param>
            <method-param>int</method-param>
            </method-params>
            </method>
            <trans-attribute>Required</trans-attribute>
         </container-transaction>
      </assembly-descriptor>
```

Runtime Behavior of the ejb-jar.xml File

The Session Bean's method **sendToSbynJms** trans-attribute is set as **Required**. The container then invokes this method with a valid transaction context. If the calling client is already associated with a transaction context, the container invokes the method within the same transactional context. If the calling client does not have an existing context, then the container automatically starts a new transactional context. The container automatically enlists all the resources managers accessed by the method.

In the sample schemas (see **Chapter 4**), the method **sendToSbynJms** is called from the **onMessage()** method in a MDB with a CMT, so this method receives the transactional context from the MDB. Our sample demonstrates that if sending a reply message to SeeBeyond JMS fails then the container rolls back the message received by the MDB. The listener port receives the same message until the maximum retry counts are reached. Once reached, the listener port then stops for user intervention on failed transaction.

2.4 e*Gate and WebSphere Application Server Messaging Modes

As discussed in **Overview of e*Gate and WebSphere Application Server Messaging** on page 12, e*Gate interacts with the WebSphere Application Server via asynchronous messaging.

The WebSphere Application Server e*Way supports the following two asynchronous messaging modes.

Session Beans publishing or sending to SeeBeyond JMS mode

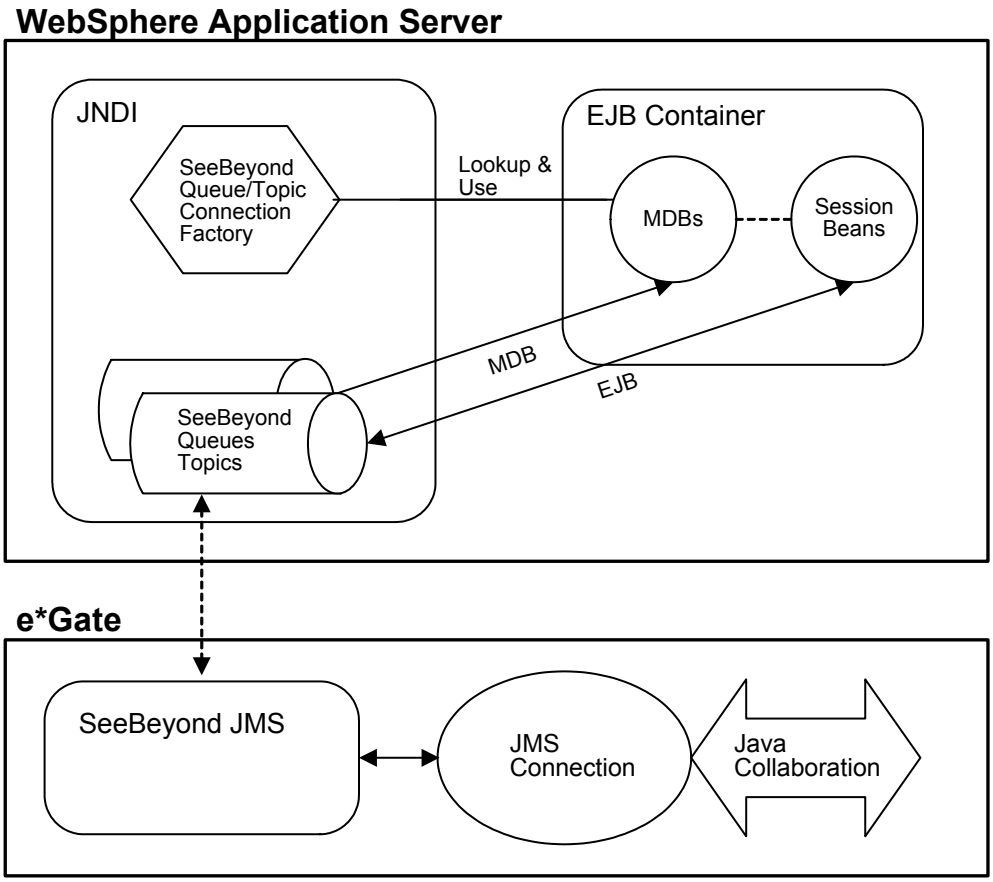
Session beans are used for publishing and sending Topic or Queue messages through SeeBeyond JMS. In this mode, the WebSphere EJBs publish to SeeBeyond JMS.

MDBs subscribing to SeeBeyond JMS mode

MDBs are used for asynchronous subscription of messages from a JMS Topic or Queue. In this mode, a message published to SeeBeyond JMS causes an MDB stored in WebSphere to execute as a result of the arriving message.

Figure 5 shows the WebSphere Application Server and e*Gate components involved in both modes of asynchronous messaging. The arrows represent message flow.

Figure 5 WebSphere Application Server and e*Gate Components



These two asynchronous messaging modes are discussed in the following section.

2.4.1. e*Gate Integration and SeeBeyond JMS

The following sections describe the WebSphere Application Server e*Way's integration with the WebSphere Application Server using the SeeBeyond implementation of JMS.

Note: *The SeeBeyond implementation of JMS is referred to as SeeBeyond JMS.*

The WebSphere Application Server e*Way incorporates SeeBeyond JMS into the WebSphere environment by registering the following SeeBeyond JMS administrative objects:

- Connection factories

- Queues
- Topics

This enables EJBs in WebSphere to receive or send messages to and from e*Gate.

In order to implement these modes, two other subsystems are used: the FSContext naming service (Sun's File System Context Provider) and the EJB container (for Session Beans and MDBs as defined in EJB 2.0).

SeeBeyond JMS

SeeBeyond JMS uses a localhost and the default port is 7555. In the sample schema described in **Chapter 4**, SeeBeyond JMS uses a localhost and port 26000. You can run the SeeBeyond JMS Server using a different host name. However, the binding name used in the program must match the external JNDI name used in the WebSphere Generic JMS provider configuration.

FSContext Naming Service

SeeBeyond JMS does not register JMS administrative objects with a naming service. It does not mandate any naming service at all. In order to integrate with WebSphere, you must use the FSContext naming service (Sun Microsystem's File System Context Provider).

FSContext is a standard JNDI source that generates a **.bindings** file in the URL you provide. You must configure WebSphere to read this file system context and bind the names into WebSphere's own name space.

The FSContext naming service allows you to bind the following SeeBeyond JMS objects:

- TopicConnectionFactory
- QueueConnectionFactory
- Topic(s)
- Queue(s)

By binding instances of these objects, any EJB can get a hold of the references to these objects by looking them up in the naming service using JNDI.

A sample stand-alone Java program is provided with the WebSphere Application Server e*Way that uses the FSContext naming service. For a complete listing of the stand-alone Java program, see **"BindJMSFactory File" on page 20**. BindJMSFactory is provided to generate a **.bindings** file in the provider URL.

SeeBeyond JMS and SeeBeyond BindJMSFactory

The program BindJMSFactory.java uses a file based naming service. BindJMSFactory uses FSContext to bind SeeBeyond JMS objects to a file-based naming service into a specified directory. You must configure the JMS Provider resource to access the specified directory in the WebSphere Administrative Console. See **External Provider URL** in **"Managing SeeBeyond JMS in the WebSphere Application Server" on page 44**.

SeeBeyond's Connection Factory and JMS topic and queue are bound to the FSContext inside the BindJMSFactory.java program. The files **compile.sh** and **runit.sh** are sample execution scripts provided to compile and build the bindings file.

The following SeeBeyond JMS ConnectionFactory objects are also bound to the FSContext inside the BindJMSFactory.java program:

- STCTopicConnectionFactory
- STCQueueConnectionFactory
- STCXATopicConnectionFactory
- STCXAQueueConnectionFactory

In addition, various JMS destination objects are bound to FSContext as well. JMS destination objects require the user to change the BindJMSFactory program to match the SeeBeyond JMS configuration of the topics and queues.

Once properly configured as a generic JMS provider in WebSphere, the bindings file can be read by WebSphere. WebSphere then binds the objects to WebSphere's namespace for the MDBs to access.

BindJMSFactory File

As already mentioned, the BindJMSFactory file is a Java program that generates bindings for several SeeBeyond JMS objects. You must compile and run the program manually to generate the bindings and write SeeBeyond JMS objects into the FSContext naming service.

The BindJMSFactory Java file is included in the WebSphere Application Server e*Way. The BindJMSFactory file is shown below.

```
public static void main(String[] args) {
        System.out.println("BindJMSFactory.main()");
        try {
            // Populate with needed properties
            Hashtable props = new Hashtable();

            /*
             *if use fscontext then the program will generate a .bindings in the PROVIDER_URL.
             *you can change the Context_PROVIDER_URL value.
             */
            if (args[0].compareToIgnoreCase("filebased") == 0 ) {
                props.put(Context.INITIAL_CONTEXT_FACTORY,
"com.sun.jndi.fscontext.RefFSContextFactory");
                props.put(Context.PROVIDER_URL, "file:C:\\websphere\\test");
            }
            else {
                /*
                 *If you have WebSphere client installed and can access directly into WebSphere's
naming service
                 *then use following initialContextfactory and url. You can change the
Context.PROVIDER_URL value
                 *to reflect where the initial context is. Refer to WebSphere naming service
document.
                 */
                props.put(Context.INITIAL_CONTEXT_FACTORY,
"com.ibm.websphere.naming.WsnInitialContextFactory");
                props.put(Context.PROVIDER_URL, "corbaloc:iiop:localhost:2809");
                props.put("java.naming.rmi.security.manager", "yes");
            };


            // Get the initial context with given properties
            jndiContext = new InitialContext(props);

            Properties factoryprops = new Properties();
           /* if you set com.seebeyond.jms.Trace.OutputFilename, it will cause enemous information
logged into the file
             * unless it is absolutely necessary, we recommend you do not do so.
             */
            //factoryprops.setProperty("com.seebeyond.jms.Trace.OutputFilename",
"C:\\eGate\\client\\logs\\wsmdbjms.log");
```

```
            /* set the properties below will cause exception right away if our jms server goes down.
              */
             factoryprops.setProperty("com.seebeyond.jms.sockets.RetryCount", "0");
             factoryprops.setProperty("com.seebeyond.jms.sockets.RetryInterval", "0");

             /*
              *Bind SeeBeyond JMS Topic objects to the naming service: fscontext or
              *websphere's naming service directly.
              */
             // you can change localhost and port number
             STCTopicConnectionFactory JMSTopicProviderFactory = new
STCTopicConnectionFactory("localhost", 26000);
             JMSTopicProviderFactory.setProperties(factoryprops);
             TopicConnectionFactory tcf = null;
             try {
                 /*you can change the jndi name here, the jndi name use here must be exactly same
                  *as in the websphere's Generic JMS Provider's configuration External JNDI Name
                  */
                 tcf = (TopicConnectionFactory)
jndiContext.lookup("jms/Topic/sbynConnectionFactory");
             } catch (Throwable e) {
                 System.out.println("jms/Topic/sbynConnectionFactory lookup exception");
                 e.printStackTrace();
                 System.out.println("jms/Topic/sbynConnectionFactory lookup exception ignored");
             }
             if (tcf == null) {
                 System.out.println("tcf is null...bind jms/Topic/sbynConnectionFactory");

                 jndiContext.bind("jms/Topic/sbynConnectionFactory", JMSTopicProviderFactory);
             } else {
                 System.out.println("tcf is NOT null...unbind then re-bind
jms/Topic/sbynConnectionFactory");

                 jndiContext.unbind("jms/Topic/sbynConnectionFactory");
                 jndiContext.rebind("jms/Topic/sbynConnectionFactory", JMSTopicProviderFactory);
             }

            /*
             *This is the JMS destination used to send a message back to SeeBeyond JMS server. You
             *can use this JMS destination inside MDB or session bean. We recomend you use the topic
            *name as websphere's Generic JMS Provider JMS destination name. Even it is not necessary
           *to do so because websphere will go to look up the jndi, and only wants the object value.
             */
             Topic topic = new STCTopic("WSToEGateTopic");
             Topic t = null;
             try {
                 /*you can change the jndi name here, the jndi name use here must be exactly same
                  *as in the websphere's Generic JMS Provider's configuration External JNDI Name
                  */
                 t = (Topic) jndiContext.lookup("jms/Topic/sbyntopicfromWS");
                 System.out.println(t.getClass().getName());
             } catch (Exception e) {
             }
             if (t == null) {
                 System.out.println("t is null...bind WSToEGateTopic as jms/Topic/sbyntopicfromWS");

                 jndiContext.bind("jms/Topic/sbyntopicfromWS", topic);
             } else {
                 System.out.println("t is NOT null...re-bind WSToEGateTopic as
jms/Topic/sbyntopicfromWS ");

                 jndiContext.unbind("jms/Topic/sbyntopicfromWS");
                 jndiContext.rebind("jms/Topic/sbyntopicfromWS", topic);
             }

             /*
             *This is the JMS destination a MDB subscribe to. You set up a Listener port to subscribe
to
             *this topic, and in your MDB deployment descriptor, you indicate the listener port name.
             *We recomend you use the topic name as websphere's Generic JMS Provider JMS destination
name.
              *Even it is not necessary to do so because websphere will go to look up the jndi, and
only
              *wants the object value.
              */
             topic = new STCTopic("eGateJMSToMDBTopic");
             t = null;
             try {
                 /*you can change the jndi name here, the jndi name use here must be exactly same
                  *as in the websphere's Generic JMS Provider's configuration External JNDI Name
                  */
                 t = (Topic) jndiContext.lookup("jms/Topic/sbyntopicFromSbynJms");
                 System.out.println(t.getClass().getName());
             } catch (Exception e) {
             }
             if (t == null) {
                 System.out.println("t is null...bind eGateJMSToMDBTopic as
jms/Topic/sbyntopicFromSbynJms");

                 jndiContext.bind("jms/Topic/sbyntopicFromSbynJms", topic);
             } else {
```

```
                System.out.println("t is NOT null...re-bind eGateJMSToMDBTopic as
jms/Topic/sbyntopicFromSbynJms");

                jndiContext.unbind("jms/Topic/sbyntopicFromSbynJms");
                jndiContext.rebind("jms/Topic/sbyntopicFromSbynJms", topic);
            }

           /*
            *Bind SeeBeyond JMS XA Topic objects to the naming service: fscontext or
            *websphere's naming service directly.
            */
            // you can change localhost and port number
            STCXATopicConnectionFactory JMSTopicProviderFactoryXA = new
STCXATopicConnectionFactory("localhost", 26000);
            JMSTopicProviderFactoryXA.setProperties(factoryprops);
            XATopicConnectionFactory xatcf = null;
            try {
                /*you can change the jndi name here, the jndi name use here must be exactly same
                 *as in the websphere's Generic JMS Provider's configuration External JNDI Name
                 */
                xatcf = (XATopicConnectionFactory)
jndiContext.lookup("jms/Topic/sbynConnectionFactoryXA");
            } catch (Throwable e) {
                System.out.println("jms/Topic/sbynConnectionFactoryXA lookup exception");
                e.printStackTrace();
                System.out.println("jms/Topic/sbynConnectionFactoryXA lookup exception ignored");
            }
            if (xatcf == null) {
                System.out.println("xatcf is null...bind");

                jndiContext.bind("jms/Topic/sbynConnectionFactoryXA", JMSTopicProviderFactoryXA);
            } else {
                System.out.println("xatcf is NOT null...unbind then re-bind");

                jndiContext.unbind("jms/Topic/sbynConnectionFactoryXA");
                jndiContext.rebind("jms/Topic/sbynConnectionFactoryXA", JMSTopicProviderFactoryXA);
            }

            topic = new STCTopic("WSToEGateTopicXA");
            t = null;
            try {
                /*you can change the jndi name here, the jndi name use here must be exactly same
                 *as in the websphere's Generic JMS Provider's configuration External JNDI Name
                 */
                t = (Topic) jndiContext.lookup("jms/Topic/sbyntopicfromWSXA");
                System.out.println(t.getClass().getName());
            } catch (Exception e) {
            }
            if (t == null) {
               System.out.println("t is null...bind WSToEGateTopicXA as jms/Topic/sbyntopicfromWSXA
");

                jndiContext.bind("jms/Topic/sbyntopicfromWSXA", topic);
            } else {
                System.out.println("t is NOT null...re-bind WSToEGateTopicXA as
jms/Topic/sbyntopicfromWSXA");

                jndiContext.unbind("jms/Topic/sbyntopicfromWSXA");
                jndiContext.rebind("jms/Topic/sbyntopicfromWSXA", topic);
            }

           /*Topic the MDB subscribe to. The MDB must have container managed transaction attributes
             *in order to use XA.
             */
            topic = new STCTopic("eGateJMSToMDBTopicXA");
            t = null;
            try {
                 /*you can change the jndi name here, the jndi name use here must be exactly same
                  *as in the websphere's Generic JMS Provider's configuration External JNDI Name
                  */
                t = (Topic) jndiContext.lookup("jms/Topic/sbyntopicFromSbynJmsXA");
                System.out.println(t.getClass().getName());
            } catch (Exception e) {
            }
            if (t == null) {
                System.out.println("t is null...bind eGateJMSToMDBTopicXA as
jms/Topic/sbyntopicFromSbynJmsXA");
                jndiContext.bind("jms/Topic/sbyntopicFromSbynJmsXA", topic);
            } else {
                System.out.println("t is NOT null...re-bind eGateJMSToMDBTopicXA as
jms/Topic/sbyntopicFromSbynJmsXA");

                jndiContext.unbind("jms/Topic/sbyntopicFromSbynJmsXA");
                jndiContext.rebind("jms/Topic/sbyntopicFromSbynJmsXA", topic);
            }


           /*
           *Bind Seebeyond JMS Queue administrative object to fscontext or websphere naming service.
            */
            STCQueueConnectionFactory JMSQueueProviderFactory = new
STCQueueConnectionFactory("localhost", 26000);
```

```
                JMSQueueProviderFactory.setProperties(factoryprops);
                QueueConnectionFactory qcf = null;
                try {
                     /*you can change the jndi name here, the jndi name use here must be exactly same
                      *as in the websphere's Generic JMS Provider's configuration External JNDI Name
                      */
                    qcf = (QueueConnectionFactory)
jndiContext.lookup("jms/Queue/sbynConnectionFactory");
                } catch (Throwable e) {
                    System.out.println("jms/Queue/sbynConnectionFactory lookup exception");
                    e.printStackTrace();
                    System.out.println("jms/Queue/sbynConnectionFactory lookup exception ignored");
                }
                if (qcf == null) {
                    System.out.println("qcf is null...bind");
                    jndiContext.bind("jms/Queue/sbynConnectionFactory", JMSQueueProviderFactory);
                } else {
                    System.out.println("qcf is NOT null...unbind then re-bind");
                    jndiContext.unbind("jms/Queue/sbynConnectionFactory");
                    jndiContext.rebind("jms/Queue/sbynConnectionFactory", JMSQueueProviderFactory);
                }
                /*Queue object used inside session bean or MDB to send a message to
                 *Seebeyond JMS queue.
                 */
                Queue queue = new STCQueue("WSToEGateQueue");
                Queue q = null;
                try {
                     /*you can change the jndi name here, the jndi name use here must be exactly same
                      *as in the websphere's Generic JMS Provider's configuration External JNDI Name
                      */
                    q = (Queue) jndiContext.lookup("jms/Queue/sbynqueuefromWS");
                    System.out.println(q.getClass().getName());
                } catch (Exception e) {
                }
                if (q == null) {
                    System.out.println("q is null...bind");
                    jndiContext.bind("jms/Queue/sbynqueuefromWS", queue);
                } else {
                    System.out.println("q is NOT null...re-bind");
                    jndiContext.unbind("jms/Queue/sbynqueuefromWS");
                    jndiContext.rebind("jms/Queue/sbynqueuefromWS", queue);
                }

                /*Queue object MDB listens to. Remember to set up the listner port is WS
                 *and configure the MDB the listner port name.
                 */
                queue = new STCQueue("eGateJMSToMDBQueue");
                q = null;
                try {
                    /*you can change the jndi name here, the jndi name use here must be exactly same
                     *as in the websphere's Generic JMS Provider's configuration External JNDI Name
                     */
                    q = (Queue) jndiContext.lookup("jms/Queue/sbynqueueFromSbynJms");
                    System.out.println(q.getClass().getName());
                } catch (Exception e) {
                }
                if (q == null) {
                    System.out.println("q is null...bind");
                    jndiContext.bind("jms/Queue/sbynqueueFromSbynJms", queue);
                } else {
                    System.out.println("q is NOT null...re-bind");
                    jndiContext.unbind("jms/Queue/sbynqueueFromSbynJms");
                    jndiContext.rebind("jms/Queue/sbynqueueFromSbynJms", queue);
                }

               /*
                *Bind Seebeyond XA Queue object, you can change the localhost and port values.
                */
                STCXAQueueConnectionFactory JMSQueueProviderFactoryXA = new
STCXAQueueConnectionFactory("localhost", 26000);
                JMSQueueProviderFactoryXA.setProperties(factoryprops);
                XAQueueConnectionFactory xaqcf = null;
                try {
                    xaqcf = (XAQueueConnectionFactory)
jndiContext.lookup("jms/Queue/sbynConnectionFactoryXA");
                } catch (Throwable e) {
                    System.out.println("jms/Queue/sbynConnectionFactoryXA lookup exception");
                    e.printStackTrace();
                    System.out.println("jms/Queue/sbynConnectionFactoryXA lookup exception ignored");
                }
                if (xaqcf == null) {
                    System.out.println("xaqcf is null...bind as jms/Queue/sbynConnectionFactoryXA");
                    jndiContext.bind("jms/Queue/sbynConnectionFactoryXA", JMSQueueProviderFactoryXA);
                } else {
                    System.out.println("qcf is NOT null...unbind then re-bind
jms/Queue/sbynConnectionFactoryXA");
                    jndiContext.unbind("jms/Queue/sbynConnectionFactoryXA");
                    jndiContext.rebind("jms/Queue/sbynConnectionFactoryXA", JMSQueueProviderFactoryXA);
                }
```

```
            /*Queue object to let session bean or MDB send a message back to Seebeyond JMS.
             */
              queue = new STCQueue("WSToEGateQueueXA");
              q = null;
              try {
                     /*you can change the jndi name here, the jndi name use here must be exactly same
                      *as in the websphere's Generic JMS Provider's configuration External JNDI Name
                      */
                  q = (Queue) jndiContext.lookup("jms/Queue/sbynqueuefromWSXA");
                  System.out.println(q.getClass().getName());
              } catch (Exception e) {
              }
              if (q == null) {
                  System.out.println("q is null...bind WSToEGateQueueXA as
jms/Queue/sbynqueuefromWSXA");
                  jndiContext.bind("jms/Queue/sbynqueuefromWSXA", queue);
              } else {
                  System.out.println("q is NOT null...re-bind WSToEGateQueueXA as
jms/Queue/sbynqueuefromWSXA");
                  jndiContext.unbind("jms/Queue/sbynqueuefromWSXA");
                  jndiContext.rebind("jms/Queue/sbynqueuefromWSXA", queue);
              }

              /*
               *Queue object MDB listens to.
               */
              queue = new STCQueue("eGateJMSToMDBQueueXA");
              q = null;
              try {
                     /*you can change the jndi name here, the jndi name use here must be exactly same
                      *as in the websphere's Generic JMS Provider's configuration External JNDI Name
                      */
                  q = (Queue) jndiContext.lookup("jms/Queue/sbynqueueFromSbynJmsXA");
                  System.out.println(q.getClass().getName());
              } catch (Exception e) {
              }
              if (q == null) {
                  System.out.println("q is null...bind");

                  jndiContext.bind("jms/Queue/sbynqueueFromSbynJmsXA", queue);
              } else {
                  System.out.println("q is NOT null...re-bind eGateJMSToMDBQueueXA as
jms/Queue/sbynqueueFromSbynJms");

                  jndiContext.unbind("jms/Queue/sbynqueueFromSbynJmsXA");
                  jndiContext.rebind("jms/Queue/sbynqueueFromSbynJmsXA", queue);
              }

          } catch (NameNotFoundException ne) {
              System.out.println("BindJMSFactory NameNotFoundException: " + ne.getExplanation());
              System.out.println(ne.getRootCause());
          } catch (Exception e) {
              System.out.println("BindJMSFactory Exception: " + e.toString());
              e.printStackTrace();
          }
      }
}
```

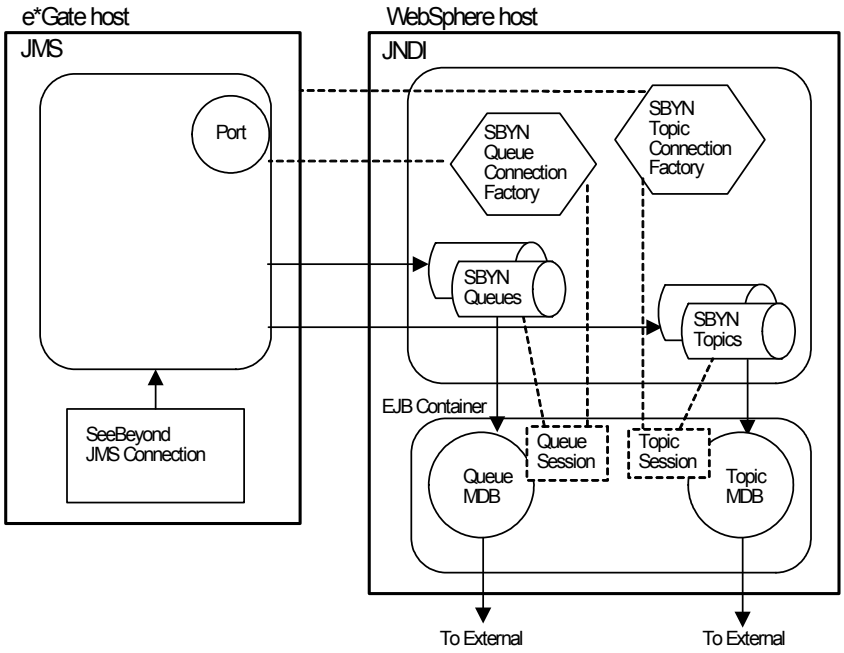
Message Flow from e*Gate to WebSphere

Once WebSphere Application Server is properly configured, the server can begin messaging. To manage the message flow from e*Gate to WebSphere, WebSphere uses the SeeBeyond TopicConnectionFactory to create the necessary JMS TopicConnection(s) and TopicSession(s). WebSphere also uses the SeeBeyond QueueConnectionFactory to create the JMS QueueConnection(s) and QueueSession(s).

Similarly, the XATopicConnectionFactory is used to create the necessary JMS XATopicConnection(s) and XATopicSession(s) and the SeeBeyond XAQueueConnectionFactory is used to create the JMS XAQueueConnection(s) and XAQueueSession(s) in MDBs using CMTs.

Figure 6 shows the components involved in messaging from e*Gate to WebSphere. The solid arrows represent message flow and the dashed lines represent associations between objects.

Figure 6 Message Flow from e*Gate to WebSphere



Once you configure and deploy the WebSphere Application Server, you can use the provided deployment descriptor to make sure the MDBs are properly configured.

EJB Deployment Descriptor ejb-jar.xml

A deployment descriptor is stored in the EJB **.jar** file. This standard EJB deployment descriptor file uses the XML markup conventions in accordance with the syntax described in the *Enterprise Java Beans 2.1 Specification*. The deployment descriptor defines EJB structural information, such as the EJB name, class, home and remote interfaces, bean type, environment entries, resource factory references, EJB references, security role references, as well as additional information based on the bean type.

The deployment descriptor demonstrates scenarios for MDBs and for Session Beans.

MDB Scenarios

The following four scenarios for MDBs are included in the deployment descriptor:

- StcJmsMDBQueue: uses normal queue connection with a bean managed transaction.
- StcJmsMDBQueueXABean: uses XA queue connection with a container managed transaction.
- StcJmsMDBTopic: uses normal topic connection with a bean managed transaction.
- StcJmsMDBTopicXABean: uses XA topic connection with a container managed transaction.

Session Bean Scenarios

The following four scenarios for Session Beans are included in the deployment descriptor:

- PubMsgToSbynJMS: used by StcJmsMDBTopic to pubish a reply under the topic name WSToEGateTopic using a bean managed transaction.
- PubMsgToSbynJMSXA: used by StcJmsMDBTopicXA to pubish a reply under the topic name WSToEGateTopicXA using a container managed transaction.
- QSendMsgToSbynJMS: used by StcJmsMDBQueue to send a reply message to Sbyn JMS under the queue name WSToEGateTopic using a bean managed transaction.
- QSendMsgToSbynJMSXA: used by StcJmsMDBQueueXA to send a reply message to SeeBeyond JMS under the queue name WSToEGateQueueXA using a container managed transaction.

The deployment descriptor is packaged in the EJB **.ear** file. The deployment descriptor is shown below.

```
<enterprise-beans>
      <message-driven id="MessageDriven_1039467765519">
            <ejb-name>StcJmsMDBQueue</ejb-name>
            <ejb-class>rchen.ws.StcJmsMDBQueueBean</ejb-class>
            <transaction-type>Bean</transaction-type>
            <acknowledge-mode>Auto-acknowledge</acknowledge-mode>
            <message-driven-destination id="MessageDrivenDestination_1040078368394">
                  <destination-type>javax.jms.Queue</destination-type>
            </message-driven-destination>
            <ejb-ref id="EjbRef_1042572354113">
                  <description>call this stateless session bean to send a reply q msg</description>
                  <ejb-ref-name>QSendMsgToSbynJMS</ejb-ref-name>
                  <ejb-ref-type>Session</ejb-ref-type>
                  <home>rchen.ws.QSendMsgToSbynJMSHome</home>
                  <remote>rchen.ws.QSendMsgToSbynJMS</remote>
                  <ejb-link>QSendMsgToSbynJMS</ejb-link>
            </ejb-ref>
      </message-driven>
      <message-driven id="MessageDriven_1039468201164">
            <ejb-name>StcJmsMDBTopic</ejb-name>
            <ejb-class>rchen.ws.StcJmsMDBTopicBean</ejb-class>
            <transaction-type>Bean</transaction-type>
            <acknowledge-mode>Auto-acknowledge</acknowledge-mode>
            <message-driven-destination id="MessageDrivenDestination_1040078368395">
                  <destination-type>javax.jms.Topic</destination-type>
                  <subscription-durability>Durable</subscription-durability>
            </message-driven-destination>
            <ejb-ref id="EjbRef_1042066078650">
                  <description>call this session bean to pub a topic msg</description>
                  <ejb-ref-name>PubMsgToSbynJMS</ejb-ref-name>
                  <ejb-ref-type>Session</ejb-ref-type>
                  <home>rchen.ws.PubMsgToSbynJMSHome</home>
                  <remote>rchen.ws.PubMsgToSbynJMS</remote>
                  <ejb-link>PubMsgToSbynJMS</ejb-link>
            </ejb-ref>
      </message-driven>
      <message-driven id="MessageDriven_1039655482969">
            <ejb-name>StcJmsMDBQueueXABean</ejb-name>
            <ejb-class>rchen.ws.StcJmsMDBQueueXABeanBean</ejb-class>
            <transaction-type>Container</transaction-type>
            <message-driven-destination id="MessageDrivenDestination_1040078368396">
                  <destination-type>javax.jms.Queue</destination-type>
            </message-driven-destination>
            <ejb-ref id="EjbRef_1042659891544">
                  <ejb-ref-name>QSendMsgToSBynJMSXA</ejb-ref-name>
                  <ejb-ref-type>Session</ejb-ref-type>
                  <home>rchen.ws.QSendMsgToSBynJMSXAHome</home>
                  <remote>rchen.ws.QSendMsgToSBynJMSXA</remote>
                  <ejb-link>QSendMsgToSBynJMSXA</ejb-link>
            </ejb-ref>
      </message-driven>
      <message-driven id="MessageDriven_1039655528475">
            <ejb-name>StcJmsMDBTopicXABean</ejb-name>
            <ejb-class>rchen.ws.StcJmsMDBTopicXABeanBean</ejb-class>
            <transaction-type>Container</transaction-type>
            <message-driven-destination id="MessageDrivenDestination_1040078368441">
                  <destination-type>javax.jms.Topic</destination-type>
                  <subscription-durability>Durable</subscription-durability>
            </message-driven-destination>
            <ejb-ref id="EjbRef_1042066353040">
                  <description>call session bean using XA</description>
                  <ejb-ref-name>PubMsgToSbynJMSXA</ejb-ref-name>
                  <ejb-ref-type>Session</ejb-ref-type>
                  <home>rchen.ws.PubMsgToSbynJMSXAHome</home>
                  <remote>rchen.ws.PubMsgToSbynJMSXA</remote>
                  <ejb-link>PubMsgToSbynJMSXA</ejb-link>
            </ejb-ref>
```

```
        </message-driven>
        <session id="PubMsgToSbynJMSXA">
              <ejb-name>PubMsgToSbynJMSXA</ejb-name>
              <home>rchen.ws.PubMsgToSbynJMSXAHome</home>
              <remote>rchen.ws.PubMsgToSbynJMSXA</remote>
              <ejb-class>rchen.ws.PubMsgToSbynJMSXABean</ejb-class>
              <session-type>Stateless</session-type>
              <transaction-type>Container</transaction-type>
              <resource-ref id="ResourceRef_1042144807763">
                    <res-ref-name>SBYN/Topic/sbynConnectionFactoryXA</res-ref-name>
                    <res-type>javax.jms.TopicConnectionFactory</res-type>
                    <res-auth>Application</res-auth>
                    <res-sharing-scope>Unshareable</res-sharing-scope>
              </resource-ref>
              <resource-ref id="ResourceRef_1042743354248">
                    <res-ref-name>JMSTopic/sbyntopicfromWSXA</res-ref-name>
                    <res-type>javax.jms.Topic</res-type>
                    <res-auth>Application</res-auth>
                    <res-sharing-scope>Unshareable</res-sharing-scope>
              </resource-ref>
        </session>
        <session id="PubMsgToSbynJMS">
              <ejb-name>PubMsgToSbynJMS</ejb-name>
              <home>rchen.ws.PubMsgToSbynJMSHome</home>
              <remote>rchen.ws.PubMsgToSbynJMS</remote>
              <ejb-class>rchen.ws.PubMsgToSbynJMSBean</ejb-class>
              <session-type>Stateless</session-type>
              <transaction-type>Bean</transaction-type>
              <resource-ref id="ResourceRef_1042144807764">
                    <res-ref-name>SBYN/Topic/sbynConnectionFactory</res-ref-name>
                    <res-type>javax.jms.TopicConnectionFactory</res-type>
                    <res-auth>Application</res-auth>
                    <res-sharing-scope>Unshareable</res-sharing-scope>
              </resource-ref>
              <resource-ref id="ResourceRef_1042743214514">
                    <description/>
                    <res-ref-name>JMSTopic/sbyntopicfromWS</res-ref-name>
                    <res-type>javax.jms.Topic</res-type>
                    <res-auth>Application</res-auth>
                    <res-sharing-scope>Unshareable</res-sharing-scope>
              </resource-ref>
        </session>
        <session id="QSendMsgToSbynJMS">
              <ejb-name>QSendMsgToSbynJMS</ejb-name>
              <home>rchen.ws.QSendMsgToSbynJMSHome</home>
              <remote>rchen.ws.QSendMsgToSbynJMS</remote>
              <ejb-class>rchen.ws.QSendMsgToSbynJMSBean</ejb-class>
              <session-type>Stateless</session-type>
              <transaction-type>Bean</transaction-type>
              <resource-ref id="ResourceRef_1042659279411">
                    <res-ref-name>SBYN/Queue/sbynConnectionFactory</res-ref-name>
                    <res-type>javax.jms.QueueConnectionFactory</res-type>
                    <res-auth>Application</res-auth>
                    <res-sharing-scope>Unshareable</res-sharing-scope>
              </resource-ref>
              <resource-ref id="ResourceRef_1042743478108">
                    <description/>
                    <res-ref-name>JMSQueue/sbynqueuefromWS</res-ref-name>
                    <res-type>javax.jms.Queue</res-type>
                    <res-auth>Application</res-auth>
                    <res-sharing-scope>Unshareable</res-sharing-scope>
              </resource-ref>
        </session>
        <session id="QSendMsgToSBynJMSXA">
              <ejb-name>QSendMsgToSBynJMSXA</ejb-name>
              <home>rchen.ws.QSendMsgToSBynJMSXAHome</home>
              <remote>rchen.ws.QSendMsgToSBynJMSXA</remote>
              <ejb-class>rchen.ws.QSendMsgToSBynJMSXABean</ejb-class>
              <session-type>Stateless</session-type>
              <transaction-type>Container</transaction-type>
              <resource-ref id="ResourceRef_1042659891564">
                    <res-ref-name>SBYN/Queue/sbynConnectionFactoryXA</res-ref-name>
                    <res-type>javax.jms.QueueConnectionFactory</res-type>
                    <res-auth>Application</res-auth>
                    <res-sharing-scope>Unshareable</res-sharing-scope>
              </resource-ref>
              <resource-ref id="ResourceRef_1042743586536">
                    <description/>
                    <res-ref-name>JMSQueue/sbynqueuefromWSXA</res-ref-name>
                    <res-type>javax.jms.Queue</res-type>
                    <res-auth>Application</res-auth>
                    <res-sharing-scope>Unshareable</res-sharing-scope>
              </resource-ref>
        </session>
  </enterprise-beans>
```

ibm-ejb-jar-bnd.xmi

The EJB deployment **.jar** file contains the **ibm-ejb-jar-bnd.xmi** file. This file specifies the listener port name, **listenerInputPortName**.

You must configure the listener port name in the WebSphere Application Server Administrative Console. For more information on configuring the see **"Configuring Listener Ports for SeeBeyond JMS" on page 49**.

The following **ibm-ejb-jar-bnd.xmi** file shows the listener port configuration of the four MDBs.

```
<ejbbnd:EJBJarBinding xmi:version="2.0" xmlns:xmi="http://www.omg.org/XMI"
xmlns:ejbbnd="ejbbnd.xmi" xmlns:ejb="ejb.xmi" xmlns:commonbnd="commonbnd.xmi"
xmlns:common="common.xmi" xmi:id="EJBJarBinding_1">
      <ejbJar href="META-INF/ejb-jar.xml#ejb-jar_ID"/>
      <ejbBindings xmi:type="ejbbnd:MessageDrivenBeanBinding"
xmi:id="MessageDrivenBeanBinding_1039468220142" listenerInputPortName="ListPortSbynJMSQueue">
            <enterpriseBean xmi:type="ejb:MessageDriven" href="META-INF/ejb-
jar.xml#MessageDriven_1039467765519"/>
            <ejbRefBindings xmi:id="EjbRefBinding_3"
jndiName="ejb/rchen/ws/QSendMsgToSbynJMSHome">
                  <bindingEjbRef href="META-INF/ejb-jar.xml#EjbRef_1042572354113"/>
            </ejbRefBindings>
      </ejbBindings>
      <ejbBindings xmi:type="ejbbnd:MessageDrivenBeanBinding"
xmi:id="MessageDrivenBeanBinding_1039468220143" listenerInputPortName="ListPortSbynJMSTopic">
            <enterpriseBean xmi:type="ejb:MessageDriven" href="META-INF/ejb-
jar.xml#MessageDriven_1039468201164"/>
            <ejbRefBindings xmi:id="EjbRefBinding_1" jndiName="ejb/rchen/ws/PubMsgToSbynJMSHome">
                  <bindingEjbRef href="META-INF/ejb-jar.xml#EjbRef_1042066078650"/>
            </ejbRefBindings>
      </ejbBindings>
      <ejbBindings xmi:type="ejbbnd:MessageDrivenBeanBinding"
xmi:id="MessageDrivenBeanBinding_1039655682770" listenerInputPortName="ListPortSbynJMSQueueXA">
            <enterpriseBean xmi:type="ejb:MessageDriven" href="META-INF/ejb-
jar.xml#MessageDriven_1039655482969"/>
            <ejbRefBindings xmi:id="EjbRefBinding_4"
jndiName="ejb/rchen/ws/QSendMsgToSBynJMSXAHome">
                  <bindingEjbRef href="META-INF/ejb-jar.xml#EjbRef_1042659891544"/>
            </ejbRefBindings>
      </ejbBindings>
      <ejbBindings xmi:type="ejbbnd:MessageDrivenBeanBinding"
xmi:id="MessageDrivenBeanBinding_1039655682771" listenerInputPortName="ListPortSbynJMSTopicXA">
            <enterpriseBean xmi:type="ejb:MessageDriven" href="META-INF/ejb-
jar.xml#MessageDriven_1039655528475"/>
            <ejbRefBindings xmi:id="EjbRefBinding_2"
jndiName="ejb/rchen/ws/PubMsgToSbynJMSXAHome">
                  <bindingEjbRef href="META-INF/ejb-jar.xml#EjbRef_1042066353040"/>
            </ejbRefBindings>
      </ejbBindings>
      <ejbBindings xmi:id="PubMsgToSbynJMSXA_Bnd" jndiName="ejb/rchen/ws/PubMsgToSbynJMSXAHome">
            <enterpriseBean xmi:type="ejb:Session" href="META-INF/ejb-jar.xml#PubMsgToSbynJMSXA"/>
            <resRefBindings xmi:id="ResourceRefBinding_11"
jndiName="jms/Topic/sbynConnectionFactoryXA">
                  <bindingResourceRef href="META-INF/ejb-jar.xml#ResourceRef_1042144807763"/>
            </resRefBindings>
            <resRefBindings xmi:id="ResourceRefBinding_1042743354248"
jndiName="jms/Topic/sbyntopicfromWSXA">
                  <bindingResourceRef href="META-INF/ejb-jar.xml#ResourceRef_1042743354248"/>
            </resRefBindings>
      </ejbBindings>
      <ejbBindings xmi:id="EnterpriseBeanBinding_1042063736475"
jndiName="ejb/rchen/ws/PubMsgToSbynJMSHome">
            <enterpriseBean xmi:type="ejb:Session" href="META-INF/ejb-jar.xml#PubMsgToSbynJMS"/>
            <resRefBindings xmi:id="ResourceRefBinding_12"
jndiName="jms/Topic/sbynConnectionFactory">
                  <bindingResourceRef href="META-INF/ejb-jar.xml#ResourceRef_1042144807764"/>
            </resRefBindings>
            <resRefBindings xmi:id="ResourceRefBinding_1042743214514"
jndiName="jms/Topic/sbyntopicfromWS">
                  <bindingResourceRef href="META-INF/ejb-jar.xml#ResourceRef_1042743214514"/>
            </resRefBindings>
      </ejbBindings>
      <ejbBindings xmi:id="EnterpriseBeanBinding_1042570847336"
jndiName="ejb/rchen/ws/QSendMsgToSbynJMSHome">
            <enterpriseBean xmi:type="ejb:Session" href="META-INF/ejb-jar.xml#QSendMsgToSbynJMS"/>
            <resRefBindings xmi:id="ResourceRefBinding_1"
jndiName="jms/Queue/sbynConnectionFactory">
                  <bindingResourceRef href="META-INF/ejb-jar.xml#ResourceRef_1042659279411"/>
            </resRefBindings>
            <resRefBindings xmi:id="ResourceRefBinding_1042743478108"
jndiName="jms/Queue/sbynqueuefromWS">
                  <bindingResourceRef href="META-INF/ejb-jar.xml#ResourceRef_1042743478108"/>
            </resRefBindings>
      </ejbBindings>
```

```
      <ejbBindings xmi:id="EnterpriseBeanBinding_1042657625956"
jndiName="ejb/rchen/ws/QSendMsgToSBynJMSXAHome">
            <enterpriseBean xmi:type="ejb:Session" href="META-INF/ejb-
jar.xml#QSendMsgToSBynJMSXA"/>
            <resRefBindings xmi:id="ResourceRefBinding_2"
jndiName="jms/Queue/sbynConnectionFactoryXA">
                  <bindingResourceRef href="META-INF/ejb-jar.xml#ResourceRef_1042659891564"/>
            </resRefBindings>
            <resRefBindings xmi:id="ResourceRefBinding_1042743586536"
jndiName="jms/Queue/sbynqueuefromWSXA">
                  <bindingResourceRef href="META-INF/ejb-jar.xml#ResourceRef_1042743586536"/>
            </resRefBindings>
      </ejbBindings>
</ejbbnd:EJBJarBinding>
```

For more information on using asynchronous messaging with the WebSphere Application Server, see the *IBM WebSphere Application Server Version 5.0 Handbook*.

Chapter 3

Configuration

This chapter provides procedures for configuring the WebSphere Application Server e*Way.

The configuration process includes three main steps:

1. Configure the IQ Manager Type.
2. Create and configure a Multi-mode e*Way, which routes and transforms data within e*Gate. See **"Multi-Mode e*Way Configuration Parameters" on page 32**.
3. Create and configure a WebSphere Application Server e*Way which enables e*Gate to communicate with the WebSphere Application Server. This step includes the following:
   A. Configure components and properties of SeeBeyond JMS. See **Configuring Components for Asynchronous Messaging Implementation using SeeBeyond JMS** on page 30.
   B. Configure the WebSphere Application Server. See **Configuring the WebSphere Application Server Components** on page 43.

Once properly configured, e*Gate Integrator uses the Multi-Mode e*Way Connection and the WebSphere Application Server e*Way Connection to send messages to the WebSphere Application Server.

3.1 Configuring Components for Asynchronous Messaging Implementation using SeeBeyond JMS

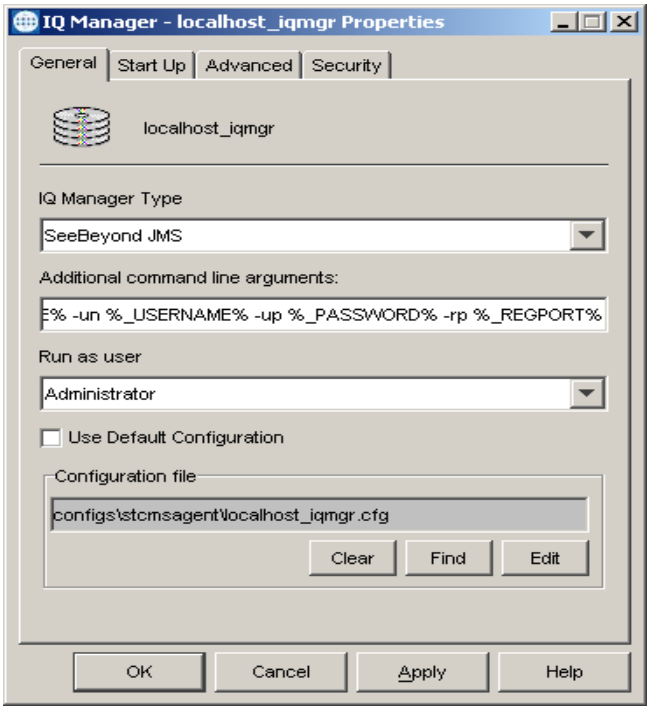
If you have not already done so, launch the Schema Designer and select a sample schema that uses SeeBeyond JMS.

3.1.1. JMS IQ Manager

Verify that the IQ Manager Type is set to **SeeBeyond JMS** (see Figure 7).

Since the WebSphere Application Server e*Way publishes Events to JMS, the IQ Manager type in your Participating Host must be set to SeeBeyond JMS.

Figure 7 SeeBeyond JMS IQ Manager



3.1.2. Multi-Mode e*Way Configuration

A Multi-Mode e*Way is a multi-threaded component used to route and transform data within e*Gate. Multi-Mode e*Ways can use multiple simultaneous e*Way Connections to communicate with external systems and Intelligent Queues (IQs).

This section provides instructions for using the e*Gate Schema Designer to create and configure a Multi-Mode e*Way.

Additional Information

This document contains basic instructions for creating and configuring a Multi-Mode e*Way. The following resources contain additional information:

- *e*Gate Integrator User's Guide*.
- *Standard e*Way Intelligent Adapter User's Guide*.
- *SeeBeyond JMS Intelligent Queue User's Guide.*
- e*Way Editor's online Help.

Creating a Multi-Mode e*Way

1. In the e*Gate Schema Designer Navigator, click the **Components** tab.
2. Open the host on which you want to create the e*Way.
3. Click the **Create a New e*Way** button.
4. Type a name for the new e*Way and click **OK**.

5 Right-click the new e*Way and select **Properties**.

The **e*Way Properties** dialog box is displayed.

6 In the **Executable File** field, select **stceway.exe** (located in the *bin* directory) if it is not selected by default.

7 In the **Additional Command Line Arguments** field, type any additional command line arguments *at the end* of the existing command-line string. Do not change any of the default arguments unless you have a specific need to do so.

8 Under the **Configuration File** field, click one of the following:

- Click **New** to create a new configuration file.
- Click **Find** to select an existing configuration file.

Note: *If a configuration file has already been assigned to this e*Way, you can edit it by clicking* ***Edit****.*

The e*Way Configuration Editor is displayed.

9 Set the parameters of the configuration file.

Note: *Configuration file parameter settings are explained in* **Multi-Mode e*Way Configuration Parameters** *on page 32.*

10 After setting the parameters, click **Save**.

11 Select **Promote to Run Time**.

12 Click **OK** to close the **e*Way Properties** dialog box.

Multi-Mode e*Way Configuration Parameters

As described in **Creating a Multi-Mode e*Way** on page 31, you can use the e*Way Configuration Editor to set the Multi-Mode e*Way configuration parameters. These parameters are described in this section.

To change Multi-Mode e*Way configuration parameters

1 In the Schema Designer's Component editor, select the e*Way you want to configure and display its properties. The Executable file for Multi-mode e*Ways is **stceway.exe**.

2 In the **Additional Command Line Arguments** box, type any additional command line arguments that the e*Way may require, taking care to insert them *at the end* of the existing command-line string. Be careful not to change any of the default arguments unless you have specific need to do so.

3 Under **Configuration File**, click **New** to create a new file, **Find** to select an existing configuration file, or **Edit** to edit the currently selected file. The Editor opens to edit settings for the Multi-Mode e*Way. The Edit Settings dialog box opens.

4 Configure the e*Way as needed for your system. Any necessary settings for a specific sample schemas are provided in **Chapter 4**.

For more information about how to use the e*Way Editor, see the e*Way Editor's online Help or the *e*Gate Integrator User's Guide*.

3.1.3 Multi-Mode e*Way Configuration Parameters

As described in **Creating a Multi-Mode e*Way** on page 31, you can use the e*Way Configuration Editor to set the Multi-Mode e*Way configuration parameters. These parameters are described in this section.

The Multi-Mode e*Way has two sets of parameters:

- **JVM Settings** on page 33
- **General Settings** on page 37

3.1.4 JVM Settings

The JVM Settings section contains the following parameters associated with the JVM (Java Virtual Machine):

- **JNI DLL Absolute Pathname** on page 33
- **CLASSPATH Prepend** on page 34
- **CLASSPATH Append From Environment Variable** on page 35
- **Initial Heap Size** on page 35
- **Maximum Heap Size** on page 35
- **Maximum Stack Size for Native Threads** on page 35
- **Maximum Stack Size for JVM Threads** on page 36
- **Disable JIT** on page 36
- **Remote debugging port number** on page 36
- **Suspend option for debugging** on page 36

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.

Required Values

A valid pathname.

Additional Information

The JNI DLL name varies on different operating system (OS) platforms, as outlined in Table 1.

Table 1 Java 2 JNI DLL Name by Operating System

Operating System	Java 2 JNI DLL Name
Windows 2000 / XP	jvm.dll
Solaris	libjvm.so
AIX	libjvm.a

The value assigned can contain a reference to an environment variable. This is done by enclosing the variable name within a pair of percent-sign (%) symbols. For example:

```
%MY_JNIDLL%
```

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

Important: *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 (Windows).*

CLASSPATH Prepend

Description

Specifies paths to be prepended to the CLASSPATH environment variable for the JVM.

Required Values

An absolute path or an environmental variable. This parameter is optional.

Additional Information

If left unset, no paths will be prepended to the CLASSPATH environment variable.

Existing environment variables may be referenced in this parameter by enclosing the variable name in a pair of percent-sign (%) symbols. For example:

```
%MY_PRECLASSPATH%
```

CLASSPATH Override

Description

Specifies the complete CLASSPATH variable to be used by the JVM. 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** and **.zip** files needed by e*Gate and the JVM must be included. It is advised that you use the **CLASSPATH Prepend** variable.*

Required Values

An absolute path or an environment variable. This parameter is optional.

Additional Information

Existing environment variables can be referenced in this parameter by enclosing the variable name in a pair of percent-sign (%) symbols. For example:

```
%MY_CLASSPATH%
```

CLASSPATH Append From Environment Variable

Description

Specifies whether the path is appended for the CLASSPATH environmental variable to **.jar** and **.zip** files needed by the JVM.

Required Values

An absolute path or an environmental variable. This parameter is optional.

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 JVM will be used.

Required Values

An integer between 0 and 2147483647. This parameter is optional.

Maximum Heap Size

Description

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

Required Values

An integer between 0 and 2147483647. This parameter is optional.

Maximum Stack Size for Native Threads

Description

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

Required Values

An integer between 0 and 2147483647. This parameter is optional.

Maximum Stack Size for JVM Threads

Description

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

Required Values

An integer between 0 and 2147483647. This parameter is optional.

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 the remote debugging of the JVM.

Required Values

An integer between 2000 and 65536.

Suspend option for debugging

Description

Specifies whether the option for debugging will be enabled or suspended upon JVM startup.

Required Values

YES or **NO**.

Auxiliary JVM Configuration File

Description

Specifies the relative path to a JVM properties file.

Required Values

A valid relative path name. This parameter is optional.

3.1.5 General Settings

The General Settings section contains the following parameters:

- **Rollback Wait Interval** on page 37
- **Standard IQ FIFO** on page 37

Rollback Wait Interval

Description

Specifies the period of time (in milliseconds) to wait before rolling back a message (retaining a message and cancelling an unsuccessful transaction).

Required Values

An integer between 0 and 99999999.

Standard IQ FIFO

Description

Specifies whether the e*Way retrieves messages from all STC-standard Intelligent Queues (IQs) in First-In-First-Out (FIFO) order

Required Values

YES or **NO**. **YES** enables FIFO. Subscribing Collaboration retrieves Events for each triggering Event Type/publishing Collaboration to which it subscribes. The Events retrieved are those with the highest priority and the oldest sequence number. The subscribing Collaboration then compares the priorities of all Events it retrieves, and among messages of the highest priority, it publishes the one with the oldest enqueue time.

Selecting **NO** disables FIFO. The Collaboration cycles through a list of triggering Event Type/publishing Collaborations to which it subscribes. The IQ Manager returns the oldest, unread Event (by sequence number) of the highest priority for that Event Type/publishing Collaboration combination. The subscribing Collaboration then processes this Event before retrieving another Event.

For more information about the Multi-Mode e*Way, see the *Standard e*Way Intelligent Adapter User's Guide*.

3.2 e*Way Connection Configuration

e*Way Connections manage the access information for specific external connections. The e*Way Connection configuration file contains the parameters necessary for connecting with the WebSphere Application Server.

You will use the e*Gate Schema Designer to create a SeeBeyond JMS e*Way Connection and set its configuration parameters, as described in this section.

3.2.1 Creating an e*Way Connection

Create and configure an e*Way Connection. The connection type should be set to "**SeeBeyond JMS**".

Note: *For the sample, the e*Way Connection is referred to as "conJMSQueueConsumer".*

Set the Event Type "get" interval to 5000.

To create and configure a SeeBeyond JMS e*Way Connection

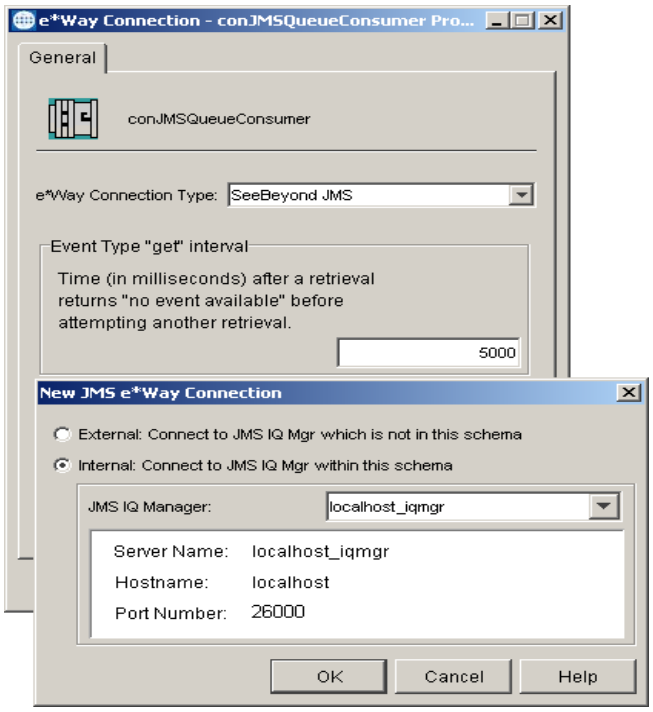
1. Activate the **Components** tab at the bottom of the Navigator (left) pane in the Schema Designer.
2. In the Navigator pane, click the **e*Way Connections** folder.
3. Click the e*Way Connection icon in the tool palette to create a new e*Way Connection.
4. In the **New e*Way Connection** dialog box, type a name (for the samples in **Chapter 4**, type the name conJMSQueueConsumer), and then click **OK** to create the **e*Way Connection**.
5. Double-click the new **e*Way Connection**. The **e*Way Connection Properties** dialog box opens.
6. From the **e*Way Connection Type** drop-down box, select **SeeBeyond JMS**. Set the Event Type "get" interval to 5000.

Click **New** under the **e*Way Connection Configuration File field**. The **New JMS e*Way Connect** dialog box opens. Indicate whether the e*Way Connection is intended for by selecting either:

- External: Connect to JMS IQ Manager which is not in the current schema
- Internal: Connect to JMS IQ Manager within this schema

If **External** is selected, you must configure the e*Way Connection, including ServerNamer, Hostname, and Port Number. If **Internal** is selected, you select a JMS IQ Manager from the drop-down, and the ServerName, Hostname, and Port Number are read in from the Registry. See Figure 8.

Figure 8 JMS e*Way Connection properties



7 Click **Edit** under the e*Way Connection Configuration File field. **The Edit Settings** dialog box opens. Enter the correct parameters for your e*Way Connection as defined in the following pages. When all parameters have been entered, from the **File** menu click **Save** and **Promote to Run Time**.

Configuring the JMS e*Way Connection Parameters

For more information about the JMS e*Way Connections, see the *SeeBeyond JMS IQ Manager User's Guide*.

This section describes the JMS e*Way configuration parameters. For SeeBeyond JMS, the e*Way Connection configuration parameters are organized into two sections:

- **General Settings** on page 39
- **Message Service** on page 41

General Settings

The General Settings control overall properties of the e*Way Connection. This section contains the following parameters:

- **Connection Type** on page 40
- **Transaction Type** on page 40
- **Delivery Mode** on page 40
- **Maximum Number of Bytes to read** on page 41
- **Default Outgoing Message Type** on page 41

- **Message Selector** on page 41
- **Factory Class Name** on page 41

Connection Type

Description

Specifies the type of connection to be established.

For publication/subscription behavior where each message is delivered to all current subscribers to the Topic, select **Topic**.

For point-to-point behavior (equivalent to "subscriber pooling" for conventional IQs) where each message is delivered to only one recipient in the pool, select **Queue.**

Required Values

Topic or **Queue**.

Transaction Type

Description

Specifies the type of transaction to be instantiated.

Important: *XA transactions for the WebSphere Application Server e*Way are managed by the WebSphere TransactionManager, NOT the e*Gate TransactionManager. For XA transactions make sure that the XAConnectionFactory(ies) are configured for the startup class.*

In **Internal** (one-phase transactional) style, a commit is necessary. The message is not saved until either a commit or a rollback is received.

In **XA-compliant** (two-phase transactional style) a two-phase commit is done: The sender sends a prepare, and the commit occurs if and only if all receivers are prepared. Collaborations that use Guaranteed Exactly Once Delivery (GEOD) of Events require XA-compliant transaction types. ***Note: This does not affect XA Transactions for the WebSphere Application Server e*Way. Read "Important" above.***

In **Non-Transactional** mode, the message is automatically saved on the server and no commit is necessary.

Required Values

Internal, **non-transactional,** or **XA-compliant**.

Delivery Mode

Description

Setting **Delivery Mode** to **Persistent** guarantees that the JMS IQ Manager stores each message safely to disk. Setting it to **Non-Persistent** does not guarantee that the message is stored safely to disk. **Non-Persistent** provides better performance but does not provide recovery.

Required Values

Non-Persistent or **Persistent**.

Important: *If the JMS IQ Manager halts when in* ***Non-Persistent*** *mode, undelivered messages are lost.*

Maximum Number of Bytes to read

Description

Your setting for this parameter depends on the size of your messages. For example, if you can anticipate that very large messages will be read, set this parameter accordingly.

Required Values

1 to **200000000.** The default is **5000.**

Default Outgoing Message Type

Description

For messages that carry no payload, or carry only a simple TextMessage payload (such as XML documents), you can set this option to **Text.**

For messages whose payload is known to be incompatible with other messaging systems, or whose payload is unknown, keep this option set to **Bytes**.

Required Values

Text. The default is **Bytes**.

Message Selector

Description

Specifies the Message Selector to be used for subscriptions.

Required Values

A string. The maximum length of query is set to 512 characters, including a null terminator.

Note: *This parameter does not check syntax. If the syntax is incorrect, the selector is ignored and the subscriber is not created.*

Factory Class Name

Description

For SeeBeyond e*Way Connections, keep the default setting: **com.stc.common.collabService.SBYNJMSFactory**

Required Values

Default: **com.stc.common.collabService.SBYNJMSFactory**

Message Service

The parameters in this section specify the low-level information required to establish the JMS. This section contains the following parameters:

- **Server Name** on page 42
- **Host Name** on page 42
- **Port Number** on page 42
- **Maximum Message Cache Size** on page 42

Server Name

Description

Specifies the name of the server (JMS IQ Manager) with which e*Gate communicates.

Required Values

A valid server name.

Host Name

Description

Specifies the name of the host on which the server (JMS IQ Manager) is running.

Required Values

A valid host name.

Port Number

Description

Specifies the port number on which the JMS IQ Manager is running.

Required Values

A valid port number between **2000** and **1000000000**.

Maximum Message Cache Size

Description

Specifies the maximum size of the message cache in bytes.

Required Values

An integer between **1** and **2147483647**.

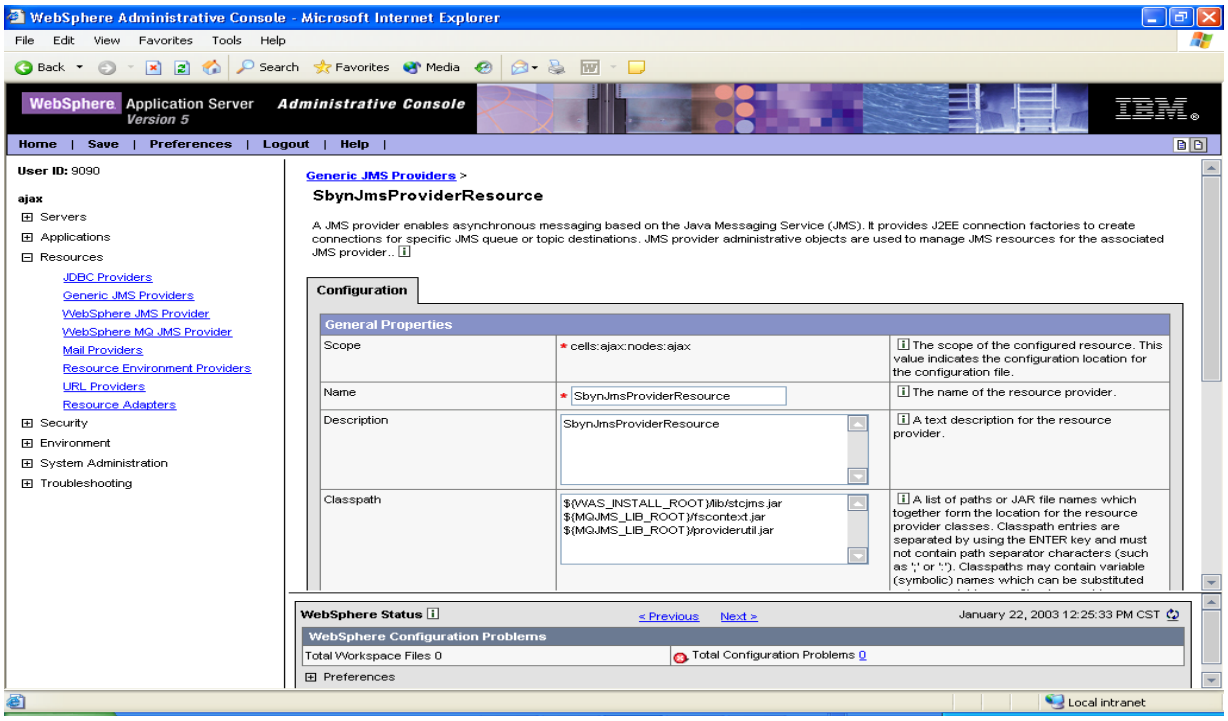
Configure the e*Way as needed for your system.

3.3 Configuring the WebSphere Application Server Components

This section describes the configuration required for sending messages between WebSphere Application Server and SeeBeyond JMS.

To use SeeBeyond JMS, you must configure the properties of SeeBeyond JMS in the WebSphere Application Server. First you must define SeeBeyond JMS in WebSphere, and then you must manage SeeBeyond JMS in the WebSphere Application Server Administrative Console, as shown in Figure 9.

Figure 9 WebSphere Application Server Administrative Console



3.3.1. Defining SeeBeyond JMS in the WebSphere Application Server

Use the administrative console to complete the following steps to define SeeBeyond JMS in the WebSphere Application Server.

1. In the navigation tree, expand **Resources** -> **Generic JMS Providers**.
2. Click **New** in the content page.
3. Define SeeBeyond JMS by specifying the appropriate values in the **General Properties** page.
4. Click **OK**.

5 To save your configuration, click **Save** on the task bar of the Administrative Console window.

3.3.2. Managing SeeBeyond JMS in the WebSphere Application Server

Use the administrative console to complete the following steps to manage SeeBeyond JMS in the WebSphere Application Server. This process configures the properties of SeeBeyond JMS in WebSphere.

1 In the navigation tree, expand **Resources**.

2 Under **Scope** select the **Node** in which the SeeBeyond JMS has been installed.

3 Click **Apply**.

4 Under General Properties configure the following properties:

A **Name**

Type a descriptive name, such as **SeeBeyond JMS**. This represents the name of the JMS provider and is for administrative purposes.

B **Description**

Type a description of **SeeBeyond JMS** for administrative purposes.

C **Classpath**

A list of paths or **.jar** file names which together form the location for the resource provider classes. You must include the following **.jar** files:

- stcjms.jar
- fscontext.jar
- providerutil.jar

Note: *After any relevant ESRs are applied, copy the latest* ***stcjms.jar*** *file from the* ***\classes*** *subdirectory in the e*Gate directory to a local destination, such as* ***$(WS_INSTALL_ROOT)/lib****. Also, the* ***fscontext.jar*** *and the* ***providerutil.jar*** *files should already be in $(MQ_INSTALL_ROOT) as part of the MQ Series (embedded messaging) installation of the WebSphere Application Server.*

D **Native Path**

An optional path to any native libraries (**.dll**, **.so**).

E External initial context factory

Type **com.sun.jndi.fscontext.RefFSContextFactory**. This is the Java classname of the initial context factory.

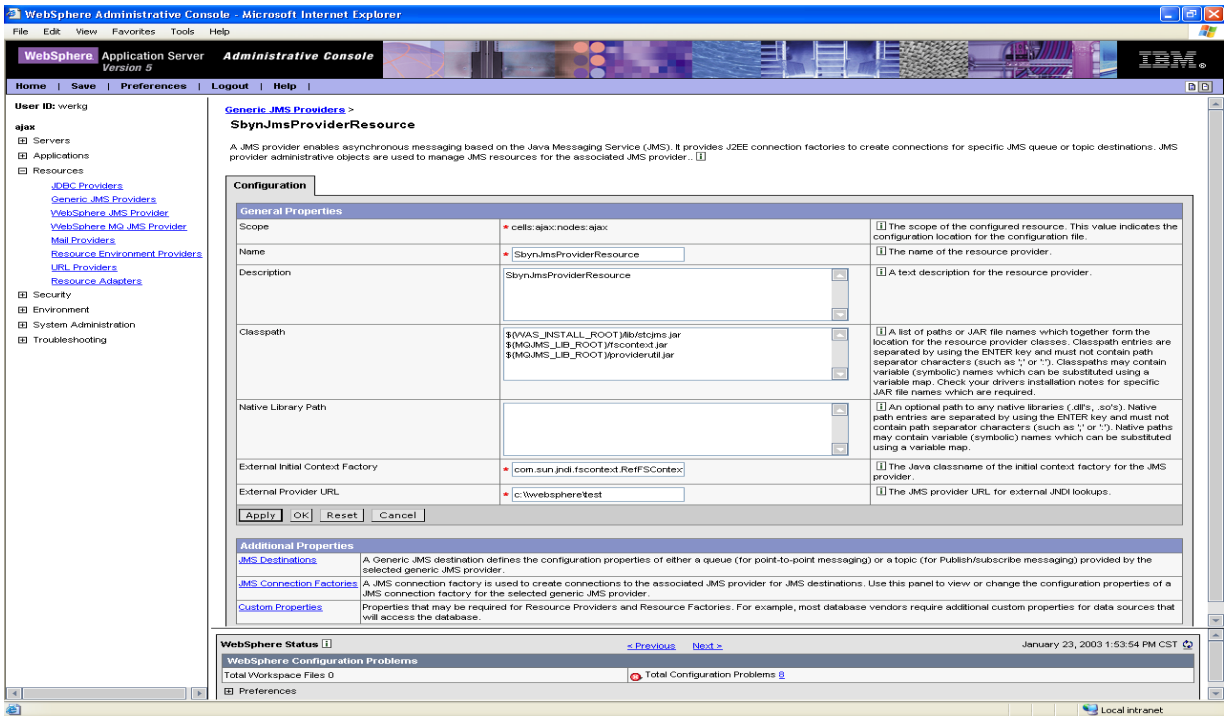
F **External provider URL**

The SeeBeyond JMS URL for external JNDI lookups. This specifies how JNDI lookups should be performed. For example, type "**`C:\\websphere\test`**".

The SeeBeyond JMS objects are bound to a file-based naming service in the directory specified in this field. For more information on this process, see **"SeeBeyond JMS and SeeBeyond BindJMSFactory" on page 19**.

Figure 10 shows the External Provider URL field in the WebSphere Application Server Administrative console. The sample directory **`C:\\websphere\test`** is shown in the field as an example of a directory specified to receive any bound JMS objects.

Figure 10 External Provider URL



Note: *Make sure that the* ***.bindings*** *file generated by the BindJMSFactory sample Java program corresponds to the directory specified in the External Provider URL field.*

5 Click **OK** or **Apply**.

6 After clicking **OK** or **Apply**, use the **Additional Properties** panel to bind the SeeBeyond JMS resource into the WebSphere JNDI namespace.

3.3.3. Configuring Resources in WebSphere for SeeBeyond JMS

Use the WebSphere administrative console to register JMS destinations and JMS connection factories into the WebSphere namespace. Using the administrative console, you must configure the following four JMS connection factories:

- queue connection factory (XA or non-XA)
- topic connection factory (XA or non-XA)
- queue destinations

- topic destinations

Configuring JMS Connection Factories for SeeBeyond JMS

The connection factories create connections with SeeBeyond JMS for a specific JMS queue or topic destination. Each connection factory contains the configuration parameters needed to create a connection to a JMS destination.

To configure properties for the four JMS connection factory objects that will connect to SeeBeyond JMS, complete the following steps.

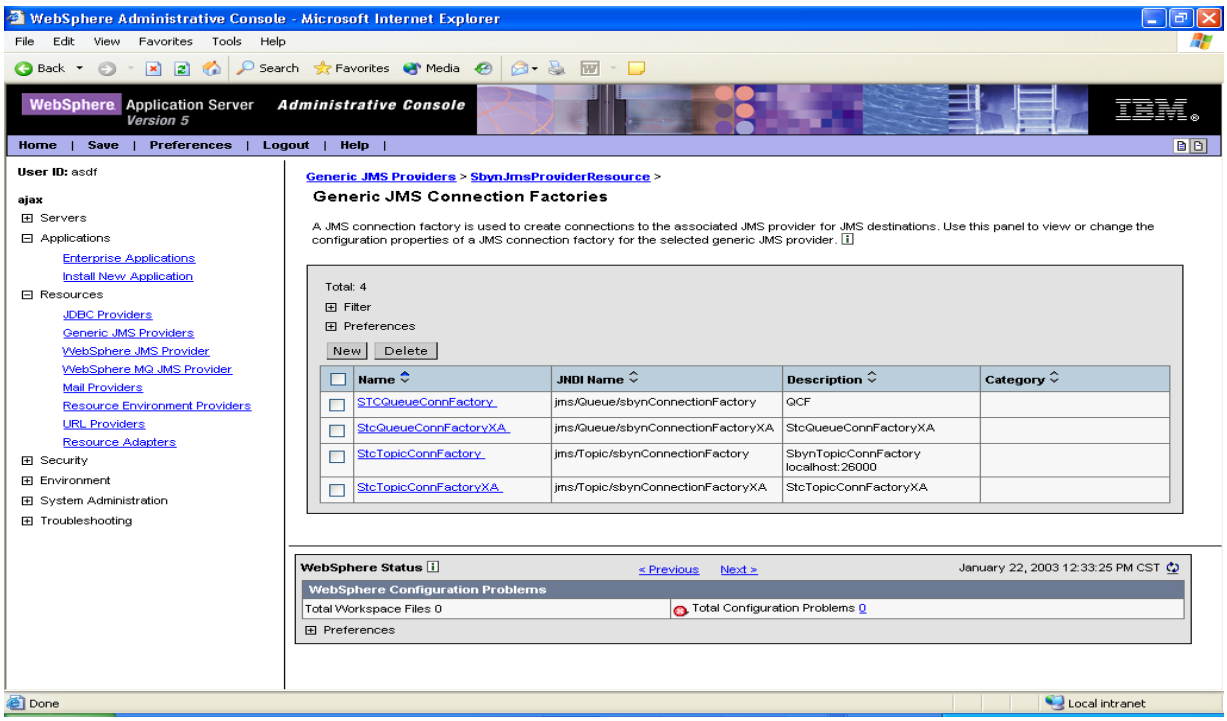
1. In the navigation tree, expand **Resources->Generic JMS Providers**.
2. Select **SeeBeyond JMS**.

   This displays a table of properties for SeeBeyond JMS and links to the types of JMS resources supported.
3. Under **Additional Properties,** click on **JMS Connection Factories**.

   This displays the SeeBeyond JMS connection factory administered objects available to application servers under the selected scope. The connection factory administered objects are used to connect to SeeBeyond JMS.

Figure 11 shows the connection factory administered objects at the selected scope.

Figure 11 SeeBeyond JMS Connection Factories



4. You must create four connection factories. To create a new connection factory object, click **New** in the content pane as shown in Figure 11.

   To change the properties of an existing connection factory, click one of the connection factories displayed.

5 The following configuration properties must be specified for each connection factory object to connect to SeeBeyond JMS.

A Name

Type a descriptive name, such as **SeeBeyond JMS**. This represents the name of the JMS provider and is for administrative purposes.

B Type

Select **QUEUE** or **TOPIC**. This represents whether the connection factory is used for point-to-point messaging (**QUEUE**) or for publish/subscribe messaging (**TOPIC**).

C JNDI Name

Type a valid JNDI name. This represents the JNDI name used to bind the connection factory into the application server's namespace.

D Description

A description of the connection factory for administrative purposes.

E Category

A category used to classify or group this connection factory, for administrative purposes.

F External JNDI Name

The connection factory JNDI name, as registered in the FSContext. WebSphere will read the FSContext under this external JNDI name and bind the same object under the JNDI name. See **"JNDI Name" on page 47**.

G User ID

The user ID used for authentication if the calling application does not provide a user ID and password.

H Password

The password used with the user ID property for authentication if the calling application does not provide a user ID and password.

6 Click **OK** once you complete configuring the queue connection factory.

7 Click **Save** to save the configuration.

To have the configuration take effect, stop then restart the application servers.

Configuring JMS Destinations for SeeBeyond JMS

A Generic JMS destination defines the configuration properties of either a queue (for point-to-point messaging) or a topic (for Publish/subscribe messaging) provided by the selected generic JMS provider. The JMS destination objects represent destinations hosted by SeeBeyond JMS.

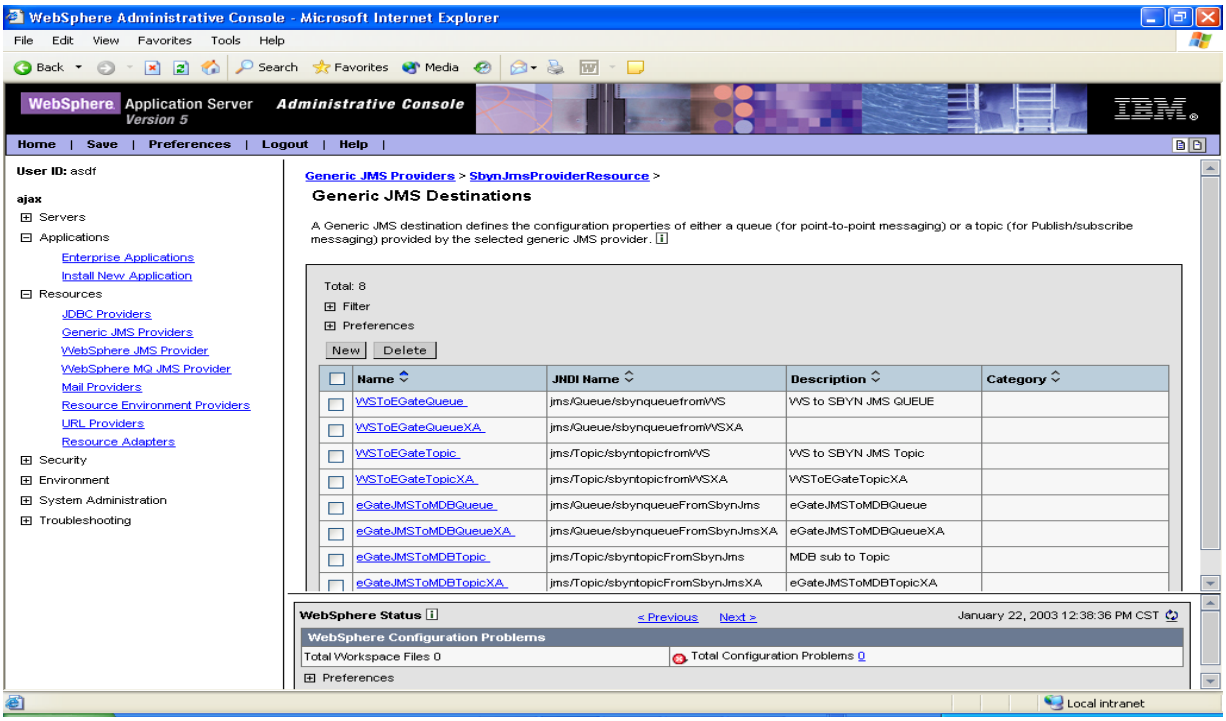
To complete the sample you must configure properties for eight JMS destinations.

To configure JMS destinations

1 In the navigation tree, expand **Resources->Generic JMS Providers ->SbynJmsProviderResource->Additional Properties ->JMS Destinations**.

Figure 12 shows the JMS destinations at the selected scope.

Figure 12 JMS Destinations



2 To create a new JMS destination, click **New**.

3 Configuration properties must be specified for a JMS destination to connect to SeeBeyond JMS. The configuration properties required for the sample are listed in Table 3.

Table 2 JMS Destination Configuration Properties

Name	JNDI name	Description
WSToEGateQueue	jms/Queue/sbynqueuefromWS	WS to SBYN JMS QUEUE s
WSToEGateQueueXA	jms/Queue/sbynqueuefromWSXA	WS to SBYN JMS QUEUE XA
WSToEGateTopic	jms/Topic/sbyntopicfromWS	WS to SBYN JMS Topic
WSToEGateTopicXA	jms/Topic/sbyntopicfromWSXA	WSToEGateTopicXA
eGateJMSToMDBQueue	jms/Queue/sbynqueueFromSbynJms	eGateJMSToMDBQueue
eGateJMSToMDBQueueXA	jms/Queue/sbynqueueFromSbynJmsXA	eGateJMSToMDBQueueXA
eGateJMSToMDBTopic	jms/Topic/sbyntopicFromSbynJms	MDB sub to Topic
eGateJMSToMDBTopicXA	jms/Topic/sbyntopicFromSbynJmsXA	eGateJMSToMDBTopicXA

A **Name**

Type the following names for each JMS destination you create.

- WSToEGateQueue
- WSToEGateQueueXA
- WSToEGateTopic
- WSToEGateTopicXA
- eGateJMSToMDBQueue
- eGateJMSToMDBQueueXA
- eGateJMSToMDBTopic
- eGateJMSToMDBTopicXA

B JNDI Name

The JNDI Name represents the JNDI name for either the queue or topic used by the JMS destination. Type the following JNDI Name for each JMS destination you create.

- jms/Queue/sbynqueuefromWS
- jms/Queue/sbynqueuefromWSXA
- jms/Topic/sbyntopicfromWS
- jms/Topic/sbyntopicfromWSXA
- jms/Queue/sbynqueueFromSbynJms
- jms/Queue/sbynqueueFromSbynJmsXA
- jms/Topic/sbyntopicFromSbynJms
- jms/Topic/sbyntopicFromSbynJmsXA

C Description

Type a description for each JMS destination you create.

4 Click **OK** once you complete configuring the JMS destinations.

5 Click **Save** to save the configuration.

To have the configuration take effect, stop then restart the application server.

Configuring Listener Ports for SeeBeyond JMS

The listener ports define the association between a connection factory, a destination, and a deployed MDB. MDBs listen upon listener ports for messages. Each listener port specifies the JMS Connection Factory and JMS Destination that a deployed MDB will listen upon.

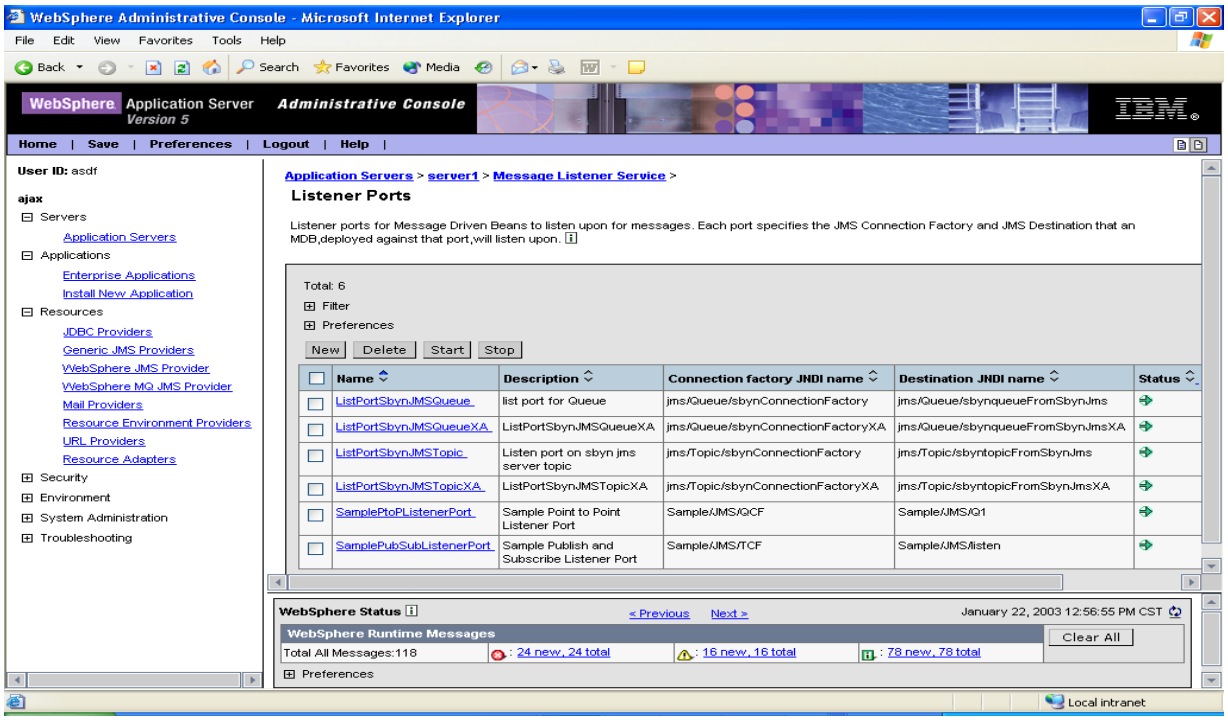
To complete the sample you must configure properties for four listener ports.

To configure listener ports

1 In the navigation tree, expand **Application Servers->server1->Message Listener Service->Listener Ports**.

Figure 13 shows the listener ports at the selected scope.

Figure 13 Listener Ports



2 To create a new listener port, click **New**.

3 Configuration properties must be specified for a listener port to connect to SeeBeyond JMS. The configuration properties required for the sample are listed in Table 3.

Table 3 Listener Port Configuration Properties

Name	Connection factory JNDI name	Destination factory JNDI name
ListPortSbynJMSQueue	jms/Queue/sbynConnectionFactory	jms/Queue/sbynqueueFromSbynJms
ListPortSbynJMSQueueXA	jms/Queue/sbynConenctionFactoryXA	jms/Queue/sbynqueueFromSbynJMSXA
ListPortSbynJMSTopic	jms/Topic/sbynConnectionFactory	jms/Topic/sbyntopicFromSbyn.Jms
ListPortSbynJMSTopicXA	jms/Topic/sbynConnectionFactoryXA	jms/Topic/sbyntopicFromSbynJmsXA

A Name

Type the following names for each listener port you create.

- ListPortSbynJMSQueue
- ListPortSbynJMSQueueXA
- ListPortSbynJMSTopic
- ListPortSbynJMSTopicXA

B Description

Type a description for each listener port you create.

C Connection factory JNDI Name

The Connection factory JNDI Name represents the JNDI name for the JMS connection factory used by the listener port. Type the following Connection Factory JNDI Name for each listener port you create.

- jms/Queue/sbynConnectionFactory
- jms/Queue/sbynConenctionFactoryXA
- jms/Topic/sbynConnectionFactory
- jms/Topic/sbynConnectionFactoryXA

D Destination JNDI Name

The Destination JNDI Name represents the JNDI name for the destination used by the listener port. Type the following Destination JNDI name for each listener port you create.

- jms/Queue/sbynqueueFromSbynJms
- jms/Queue/sbynqueueFromSbynJMSXA
- jms/Topic/sbyntopicFromSbyn.Jms
- jms/Topic/sbyntopicFromSbynJmsXA

4 Click **OK** once you complete configuring the listener port.

5 Click **Save** to save the configuration.

To have the configuration take effect, stop then restart the application server.

Chapter 4

Implementation

This chapter contains basic information for implementing the WebSphere Application Server e*Way in a production environment. The sample schemas that are included with the e*Way are intended to demonstrate and provide a comprehensive overview of the e*Way implementation process.

This chapter provides detailed information for creating and configuring the necessary components for the WebSphere Application Server e*Way sample schemas. Review this information carefully, as it illustrates how to implement the WebSphere Application Server e*Way in your own production environment.

4.1 Implementation Process Overview

The WebSphere Application Server e*Way enables e*Gate to connect with the WebSphere Application Server. When the WebSphere Application Server e*Way is installed with e*Gate Integrator, you can create and configure schemas using the e*Gate Schema Designer.

A schema contains the parameters for components that control, route, and transform data as it moves through e*Gate in a predefined system configuration. For the most part, the steps involved in creating an e*Way have already been implemented for the imported sample schemas.

Implementing the WebSphere Application Server e*Way

To implement the WebSphere Application Server e*Way within an e*Gate system requires the following:

- Install the WebSphere Application Server e*Way.
- Create one or more e*Way components and configure the properties and parameters.
- Define the necessary e*Way Connections and configure the properties and parameters.
- Define Collaboration Rules to extract selected information from a source Event and process it according to the Collaboration Service associated with the Collaboration Rules.
- Define Collaborations to receive and process Event Types and then forward the output to other e*Gate components.

- Configure any other components necessary to complete the schema.
- Test the schema and make any necessary adjustments.

This chapter describes how to perform each of these steps using various sample schemas as demonstration tools.

Note: *For additional information on creating or modifying any component within the e*Gate Schema Designer, see the e*Gate Schema Designer's online Help.*

4.2 Preparing for the Sample Schemas

The following pages contain sample schemas that explain how the components for the WebSphere Application Server e*Way are implemented.

The section, **"Creating the Sample Schemas" on page 54** describes the various sample schemas for the WebSphere Application Server e*Way available on the installation CD-ROM.

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 the e*Gate Schema Designer GUI is installed.

Note: *For more information about creating or modifying any component within the e*Gate Schema Designer, see the e*Gate Schema Designer's online Help.*

4.2.1. Considerations

- XA transactions for the WebSphere Application Server e*Way are managed by the WebSphere TransactionManager, NOT the e*Gate TransactionManager or in the e*Way Connection parameters. For XA transactions make sure that the XAConnectionFactory is configured in the BindJMSFactory class.
- Any third-party J2EE-compliant Integrated Development Environment (IDE) tool can be used to create the application **.jar** file.

4.2.2. Preparing for Implementing the Sample Schemas

Before creating one or more of the sample schemas you must do the following to enable asynchronous messaging between WebSphere EJB and SeeBeyond JMS.

Step 1: Configure WebSphere to create JNDI entries for SeeBeyond JMS

Configure WebSphere to create JNDI entries in the directory service for SeeBeyond JMS on WebSphere Application Server instance startup. For more information, see **"Configuring the WebSphere Application Server Components" on page 43**.

Step 2: Create new message driven beans

Use the WebSphere Studio Application Developer to build EJBs, or use the **.jar** file provided.

Step 3: Packaging and deployment

Use the WebSphere Application Assembly Tool to package the EJB for deployment to other applications. The WebSphere Application Assembly Tool is provided in the WebSphere Application Server installation.

Step 4: Install the EJB in WebSphere

Import the already created EJB **.jar** file into WebSphere Application Assembly Tool. This step generates the EJB **.ear** file.

Note: *While preparing for the application installation, make sure that the* ***Override existing bindings*** *option is selected. The* ***.bindings*** *file generated by the sample Java program BindJMSFactory should be selected for the specific* ***.bindings*** *file field to overwrite the default binding.*

4.3 Creating the Sample Schemas

Sample schemas for the WebSphere Application Server e*Way asynchronous (JMS) implementation is available in the **..\samples\ewwebsphere\java-WAS5.1** folder of the installation CD-ROM. Import the **websphere_sample.zip** file into e*Gate to create the following:

- **WebSphere Sample Schemas** on page 55 contains the four samples listed below that demonstrate WebSphere Application Server e*Way asynchronous messaging using the SeeBeyond JMS Connection. To install the JMSAsynchProducersConsumer sample import **JMSAsynchProducersConsumer.zip** into e*Gate.
  - **JMSQueueSend and JMSQueueReceive Sample** on page 55 demonstrates asynchronous messaging from e*Gate to WebSphere via the SeeBeyond JMS Queue. The e*Way picks up a message and publishes it to the SeeBeyond JMS Queue. The message is then subscribed to by the WebSphere MDB.
  - **JMSXAQueueSend Sample** on page 63 demonstrates asynchronous interaction with an XA transaction. The e*Way picks up a message and publishes it to the SeeBeyond JMS Queue. The message is then subscribed to by the WebSphere XA MDB.
  - **JMSTopicPublish and JMSTopicSubscribe Sample** on page 66 demonstrates asynchronous messaging in which the e*Way picks up a message from a file and publishes it to the SeeBeyond JMS Topic where the message is subscribed to by the WebSphere MDB.
  - **JMSXATopicPublish Sample** on page 74 demonstrates asynchronous interaction with an XA transaction. The e*Way publishes to an XA JMS Topic which is subscribed to by a WebSphere MDB.

4.3.1. Installing a Sample Schema

Once you have imported the **.zip** files in to e*Gate, you must import the schema at the startup of the e*Gate Schema Designer, or select **New Schema** from the File menu of the e*Gate Enterprise. For either case, select **Create from export** and navigate to the **.zip** file containing the sample.

4.4 WebSphere Sample Schemas

The WebSphere sample schemas contains four e*Ways configured to utilize the SeeBeyond JMS Connection. The e*Ways deliver and receive messages to and from the Enterprise JavaBeans running inside the WebSphere container. The schemas also configure the IQ Manager as a SeeBeyond JMS IQ Manager. There are two modes of operation occurring in the sample schemas: e*Ways sending or publishing messages to a Queue or Topic, and e*Ways which receive or subscribe to a Queue or a Topic.

4.4.1. Setting up the WebSphere Schemas

When setting up the WebSphere schemas, do the following:

1. For directions on importing the sample see **Installing a Sample Schema** on page 54.
2. Make sure that the sample schema is running first prior to deploying the EJBs. This ensures that the SeeBeyond IQ Manager (SeeBeyond JMS Server) is available so that the WebSphere container can create the connections on behalf of the MDBs during deployment.
3. Do NOT feed messages into the feeder e*Ways UNTIL the sample EJBs are deployed. This guarantees that there are subscriber or receiver MDBs running before messages are sent to Topics or Queues.
4. Start the WebSphere server.
5. Deploy the sample EJBs.
6. Feed messages to the feeder e*Ways. Messages are then published to SeeBeyond JMS. The MDB subscribes to and receives the messages to the EJBs. The messages are then visible in the WebSphere **SystemOut.log file**. MDBs process the messages and publish them back to SeeBeyond JMS. Finally, the eater e*Ways get the messages from JMS and the messages are then written to files.

4.4.2. JMSQueueSend and JMSQueueReceive Sample

JMSQueueSend e*Way

In this sample, the JMSQueueSend e*Way (**stcewfile.exe**) acts as a feeder of messages to the eGateJMSToMDBQueue queue. The JMSQueueSend e*Way looks for files with extension **.qfin** as input files (the input directory configured is **C:\InputData**). The colJMSQueueSend Collaboration subscribes to external (for an event from a file) and publishes to the conJMSQueueProducer JMS Connection. The conJMSQueueProducer

JMS Connection is configured to use the internal SeeBeyond JMS IQ Manager as the JMS server. The colJMSQueueSend Collaboration uses the crJMSQueueSend Collaboration Rule which copies data from the source event to the output event.

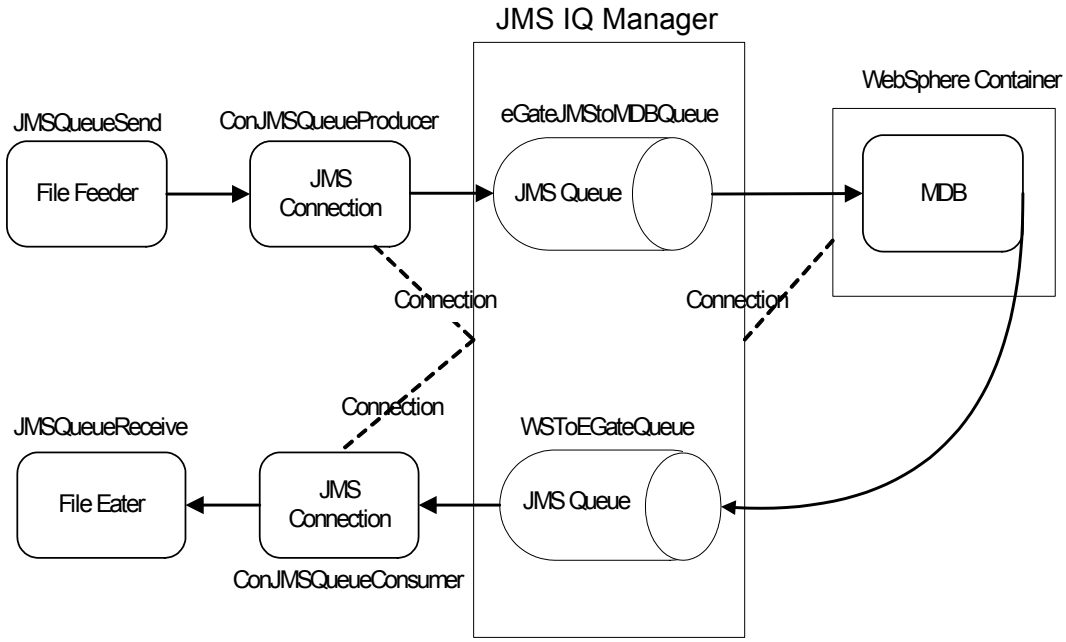
JMSQueueReceive e*Way

The JMSQueueReceive e*Way (stcewfile.exe) acts as a eater of messages coming from the WSToEGateQueue queue. The colJMSQueueReceive Collaboration subscribes to the conJMSQueueConsumer JMS Connection on the WSToEGateQueue queue. The conJMSQueueConsumer JMS Connection is configured to use the internal SeeBeyond JMS IQ Manager as the JMS server. The colJMSQueueReceive Collaboration uses the crJMSQueueReceive Collaboration Rule which displays the message received to standard output and publishes the message to the external.

JMSQueueSend and JMSQueueReceive Sample Message Flow

Figure 14 shows the components of the JMSQueueSend and JMSQueueReceive sample.

Figure 14 JMSQueueSend and JMSQueueReceive Sample Components



As shown in Figure 14, the File Feeder reads a file containing the input message event. A feeder Collaboration subscribes from external and publishes the input message as a eGateJMSToMDBQueue event to the JMS Connection. The JMS Connection is configured to use a JMS Queue and acts as a QueueSender. Both the JMS Connection and the MDB are configured to connect to the JMS IQ Manager as the JMS server. The MDB receives messages from the eGateJMSToMDBQueue queue and displays the message it receives to the WebSphere console. The MDB sends the message to the WSToEGateQueue. The file eater receives messages from the WSToEGateQueue and writes it to an output file.

Configuring the JMSQueueSend and JMSQueueReceive Sample

Once the sample has been successfully imported into e*Gate, you must configure it to correspond to the system as necessary.

Configuring the e*Ways

Each of the configuration files associated with the e*Way must be configured as needed, saved, and promoted to run time. Specifically, the parameters listed in Table 4 must be configured as shown.

Table 4 e*Way Configuration Parameters - JMSQueueSend

e*Way Configuration Parameters	
General Settings - Set as directed, otherwise leave as default.	
AllowIncoming	YES
AllowOutgoing	NO
PerformanceTesting	NO
Outbound (send) settings - Set as directed, otherwise leave as default.	
OutputDirectory	C:\DATA
OutputFileName	output%d.dat
MultipleRecordsPerFile	YES
MaxRecordsPerFile	10000
AddEOL	Yes
Poller (inbound) settings - Set as directed, otherwise leave as default.	
PollDirectory	C:\INDATA
InputFileMask	*.qfin
PollMilliseconds	1000
RemoveEOL	YES
MultipleRecordsPerFile	NO
MaxBytesPerLine	4096
BytesPerLineIsFixed	NO
File Records Per eGate Event	1
Performance Testing - Set as directed, otherwise leave as default.	
Performance Testing	100
InboundDuplicates	1

Table 5 e*Way Configuration Parameters - JMSQueueReceive

e*Way Configuration Parameters	
General Settings - Set as directed, otherwise leave as default.	
AllowIncoming	NO
AllowOutgoing	YES
PerformanceTesting	NO
Outbound (send) settings - Set as directed, otherwise leave as default.	

e*Way Configuration Parameters	
OutputDirectory	C:\DATA
OutputFileName	queuereceive%d.dat
MultipleRecordsPerFile	YES
MaxRecordsPerFile	10000
AddEOL	Yes
Poller (inbound) settings - See Table 4 on page 57.	
Performance Testing - See Table 4 on page 57.	

Configuring the e*Way Connection

Each of the configuration files associated with the e*Way must be configured as needed, saved, and promoted to run time. Specifically, the parameters listed in Table 6 must be configured as shown.

The conJMSQueueConsumer and conJMSQueueProducer e*Way Connection parameters associated with the JMSQueueSend and JMSQueueReceive sample appear as shown in Table 6.

Table 6 e*Way Connection Parameters - JMSQueueSend and JMSQueueReceive

e*Way Connection Parameters	
General Settings - Set as directed, otherwise leave as default.	
Connection Type	Queue
Transaction Type	Internal
SDelivery Mode	Persistent
Maximum Number of Bytes to read	10000000
Default Outgoing Message Type	Text
Message Selector	
Factory Class Name	com.stc.commonService.SBYNJMSFactory
Message Service - Set as directed, otherwise leave as default.	
Server Name	localhost_iqmgr
Host Name	localhost
Port Number	26000
Maximum Message Cache Size	100

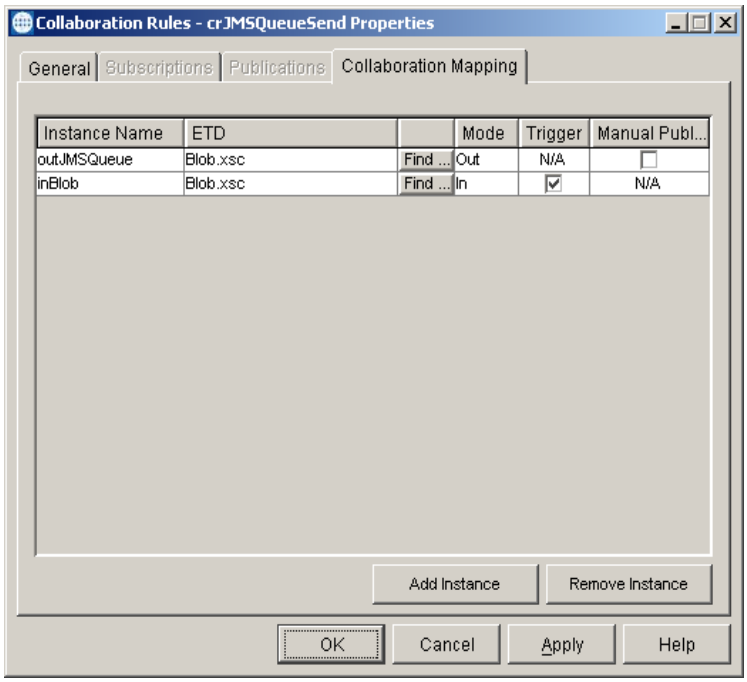
For more information on e*Way Connection Configuration Parameters for JMS see **Configuring the JMS e*Way Connection Parameters** on page 39.

Creating Collaborations

JMSQueueSend Collaboration Rule Mapping

The Collaboration Mapping associated with the crJMSQueueSend Collaboration Rule appears as shown in Figure 15.

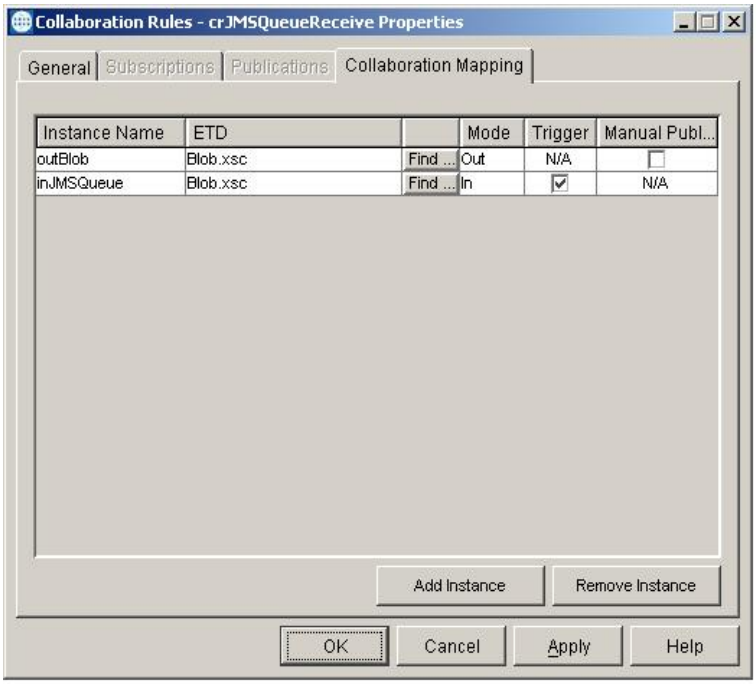
Figure 15 crJMSQueueSend - Collaboration Mapping



JMSQueueReceive Collaboration Rule Mapping

The Collaboration Mapping associated with the crJMSQueueReceive Collaboration Rule appears as shown in Figure 16.

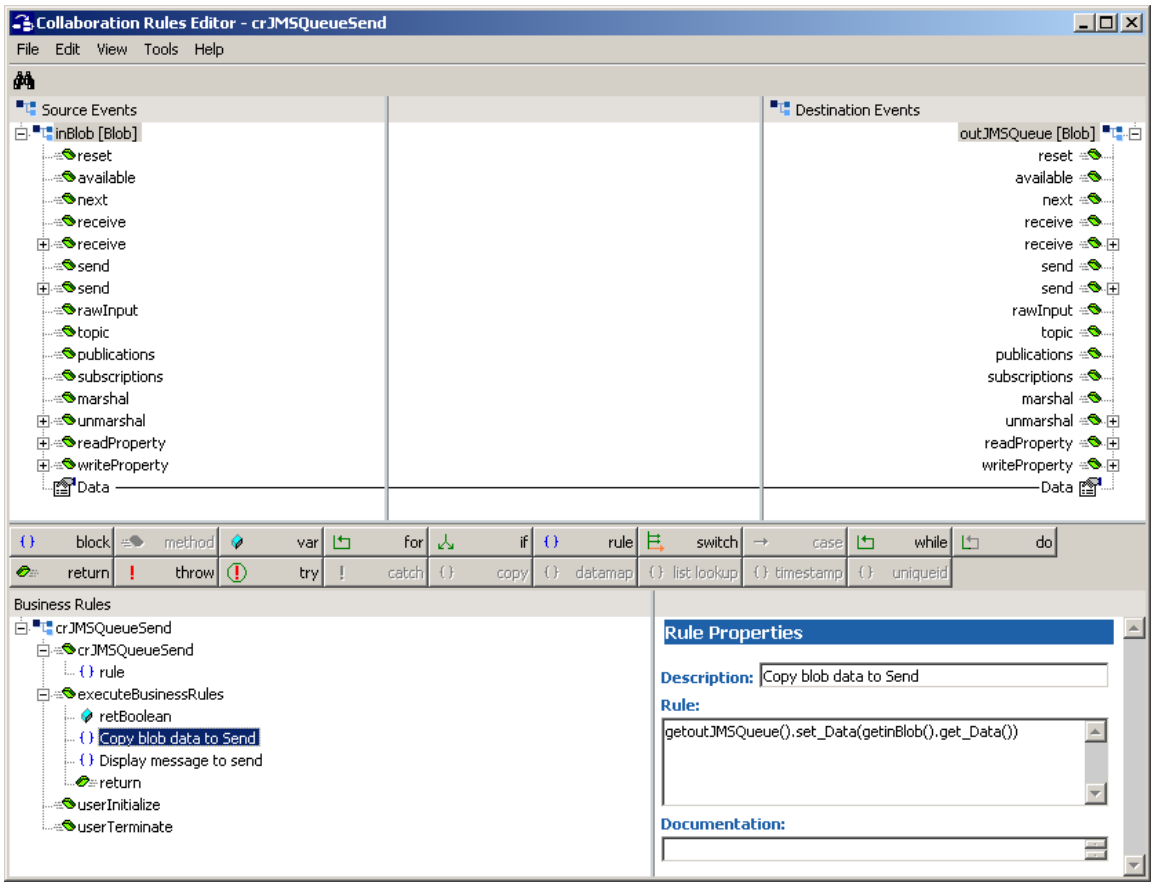
Figure 16 crJMSQueueReceive - Collaboration Mapping



JMSQueueSend Collaboration Rules

The crJMSQueueSend Collaboration Rules appear as shown in Figure 17.

Figure 17 Collaboration Rules - crJMSQueueSend



Each new rule is created by clicking the **rule** button on the Business Rules toolbar. For additional information on using the Java Collaboration Rules Editor, see the *e*Gate Integrator User's Guide*. The crJMSQueueSend business rules are created as follows:

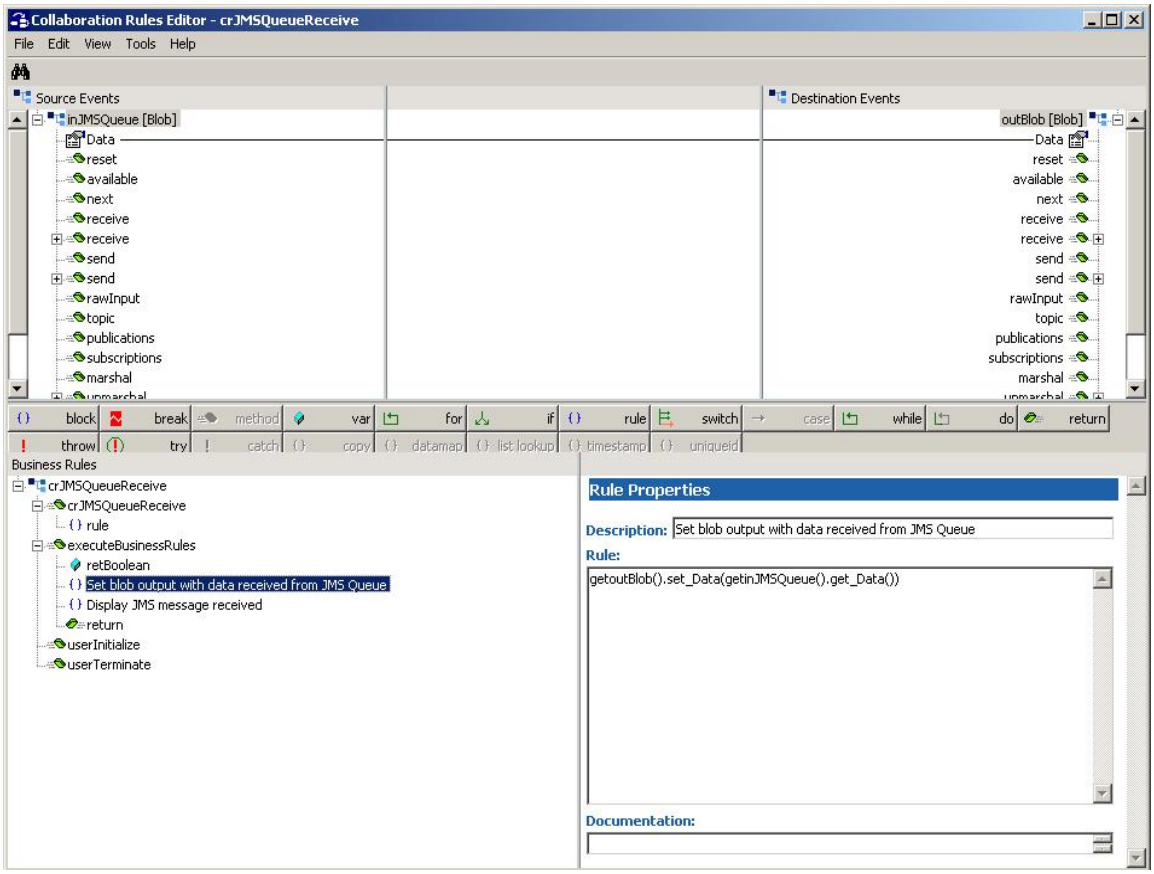
1. "**Copy blob data to Send**" is created by dragging **Data** located under Source Events command node and dropping it on **Data** located under the Destination Events.
2. "**Display message to send**" is created by dragging **Data** located under Source Events command node into the Rule Properties, Rules window and entering code before and after to create the following code:

```
System.out.println("\nSending Message:\n*****Start of Message*****\n" + getinBlob().get_Data()
+ "\n*****End of Message*****\n")
```

JMSQueueReceive Collaboration Rules

The crJMSQueueReceive Collaboration Rules appear as shown in Figure 18.

Figure 18 Collaboration Rules - crJMSQueueReceive



Each new rule is created by clicking the **rule** button on the Business Rules toolbar. For additional information on using the Java Collaboration Rules Editor, see the *e*Gate Integrator User's Guide*. The crJMSQueueReceive business rules are created as follows:

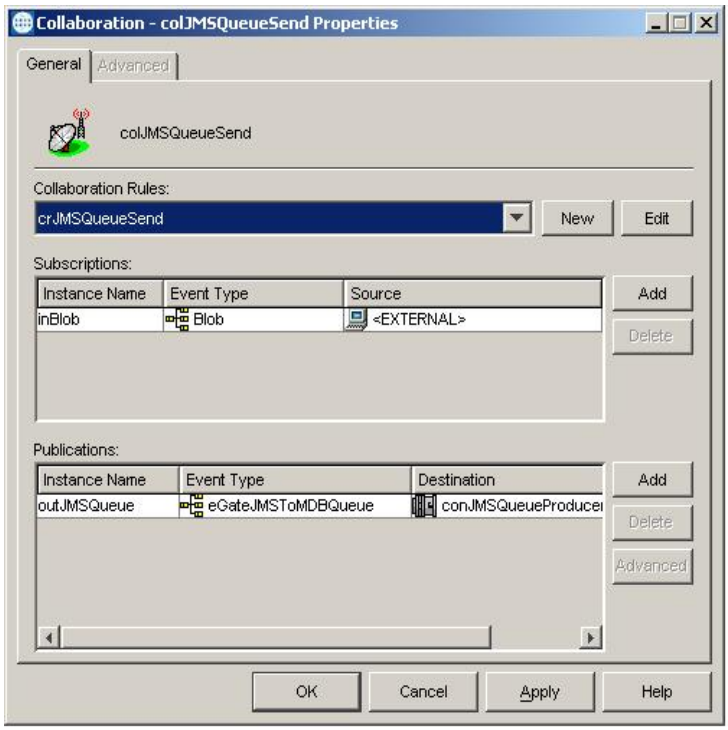
1. "**Copy blob data to Send**" is created by dragging **Data** located under Source Events command node and dropping it on **Data** located under the Destination Events.
2. "**Display message to send**" is created by dragging **Data** located under Source Events command node into the Rule Properties, Rules window and entering code before and after to create the following code:

```
System.out.println("\nSending Message:\n*****Start of Message*****\n" + getinBlob().get_Data()
+ "\n*****End of Message*****\n")
```

JMSQueueSend Collaboration Properties

The colJMSQueueSend Collaboration Properties for the JMSQueueSend sample appears as shown in Figure 19.

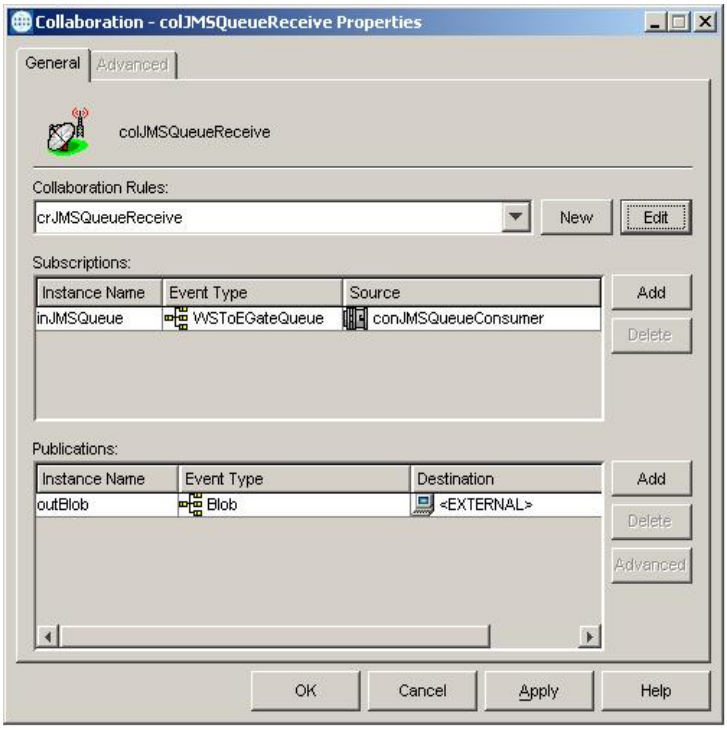
Figure 19 colJMSQueueSend - Collaboration Properties



JMSQueueReceive Collaboration Properties

The colJMSQueueReceive Collaboration Properties for the JMSQueueReceive sample appear as shown in Figure 20.

Figure 20 colJMSQueueReceive - Collaboration Properties



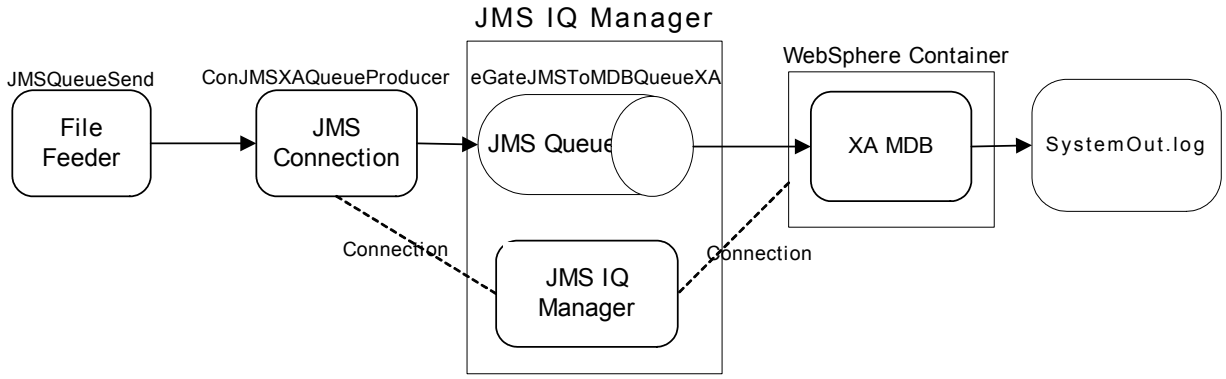
4.4.3. JMSXAQueueSend Sample

In this sample, the JMSXAQueueSend e*Way (stcewfile.exe) acts as a feeder of messages to the eGateJMSToMDBQueueXA queue. The JMSXAQueueSend e*Way looks for files with the extension **.xaqfin** as input files (the input directory configured is **C:\INDATA**). The colJMSXAQueueSend Collaboration subscribes to external (for an event from a file) and publishes to the conJMSXAQueueProducer JMS Connection. The conJMSXAQueueProducer JMS Connection Point is configured to use the internal SeeBeyond JMS IQ Manager as the JMS server. The colJMSXAQueueSend Collaboration uses the crJMSQueueSend Collaboration Rule which copies data from the source event to the output event.

JMSXAQueueSend e*Way

Figure 21 shows the components of the JMSXAQueueSend sample.

Figure 21 JMSXAQueueSend Sample Components



As shown in Figure 21, the File Feeder reads a file containing the input message event. A feeder Collaboration subscribes from external and publishes the input message to the JMS Connection as a eGateJMSToMDBQueueXA event. The JMS Connection is configured to use a JMS Queue and therefore acts as a QueueSender. Both the JMS Connection and the MDB are configured to connect to the JMS IQ Manager as the JMS server. The MDB receives the message and prints to the **SystemOut.log** file in **WebSphere/logs/<servername>**.

Configuring the JMSXAQueueSend Sample

Once the sample has been successfully imported into e*Gate, you must configure it to correspond to the information as necessary.

Configuring the e*Ways

Each of the configuration files associated with the e*Way must be configured as needed, saved, and promoted to run time.

The JMSXAQueueSend **e*Way Connection Configuration settings** are the same as those in **Table 6 on page 58**.

Configuring the e*Way Connection

Each of the configuration files associated with the e*Way must be configured as needed, saved, and promoted to run time.

Configuration parameters for the **conJMSXAQueueProducer** e*Way Connection used with the JMSXAQueueSend sample are the same as those in **Table 4 on page 57** with the exception of those shown in Table 7.

Table 7 e*Way Connection Parameters - JMSXAQueueProducer

e*Way Connection Parameters	
General Settings - Set as directed, otherwise leave as default.	
Connection Type	Queue
Transaction Type	XA-Compliant
SDelivery Mode	Persistent

Table 7 e*Way Connection Parameters - JMSXAQueueProducer

e*Way Connection Parameters	
Maximum Number of Bytes to read	10000000
Default Outgoing Message Type	Text
Message Selector	
Factory Class Name	com.stc.commonService.SBYNJMSFactory
Message Service - Set as directed, otherwise leave as default.	
Server Name	localhost_iqmgr
Host Name	localhost
Port Number	26000
Maximum Message Cache Size	100

- For more information on e*Way Connection Configuration Parameters for JMS see **Configuring the JMS e*Way Connection Parameters** on page 39.

Creating Collaborations

JMSXAQueueSend Collaboration Rule Mapping

The JMSXAQueueSend sample uses the crJMSQueueSend Collaboration Rule Mapping (see **crJMSQueueSend - Collaboration Mapping** on page 59).

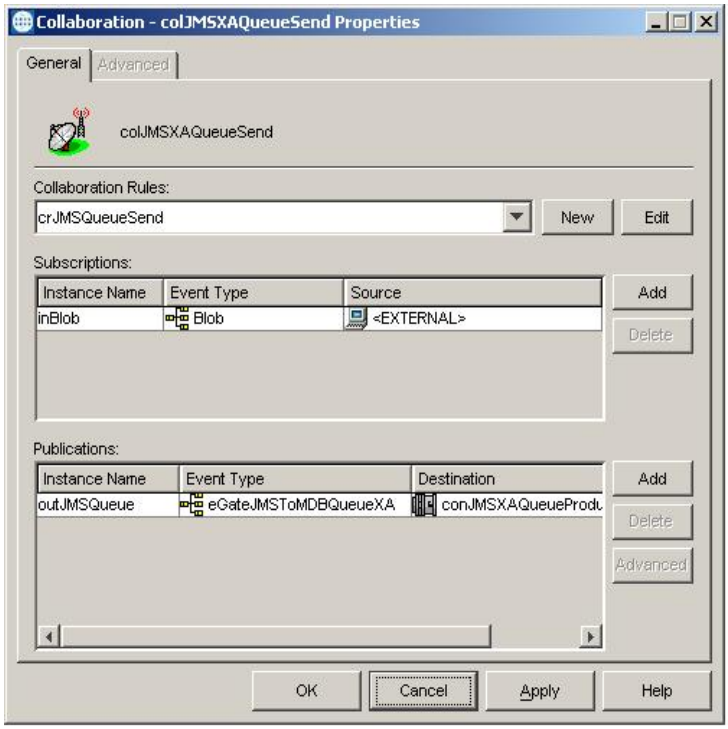
JMSXAQueueSend Collaboration Rules

The JMSXAQueueSend sample uses the JMSQueueSend Collaboration Rules (see **Collaboration Rules - crJMSQueueSend** on page 60).

JMSXAQueueSend Collaboration Properties

The colJMSXAQueueSend Collaboration for the JMSXAQueueSend sample appears as follows in Figure 22.

Figure 22 colJMSXAQueueSend - Collaboration Properties



4.4.4. JMSTopicPublish and JMSTopicSubscribe Sample

JMSTopicPublish e*Way

In this sample, the JMSTopicPublish e*Way (stcewfile.exe) acts as a feeder of messages to the eGateJMSToMDBTopic topic. The JMSTopicPublish e*Way looks for files with the extension **.tfin** as input files (the configured input directory is C:\INDATA). The colJMSTopicPublish Collaboration subscribes to external (for an event from a file) and publishes to the conJMSTopicProducer JMS Connection. The conJMSTopicProducer JMS Connection is configured to use the internal SeeBeyond JMS IQ Manager as the JMS server. The colJMSTopicPublish Collaboration uses the crJMSTopicPublish Collaboration Rule which simply copies data from the source event to the output event.

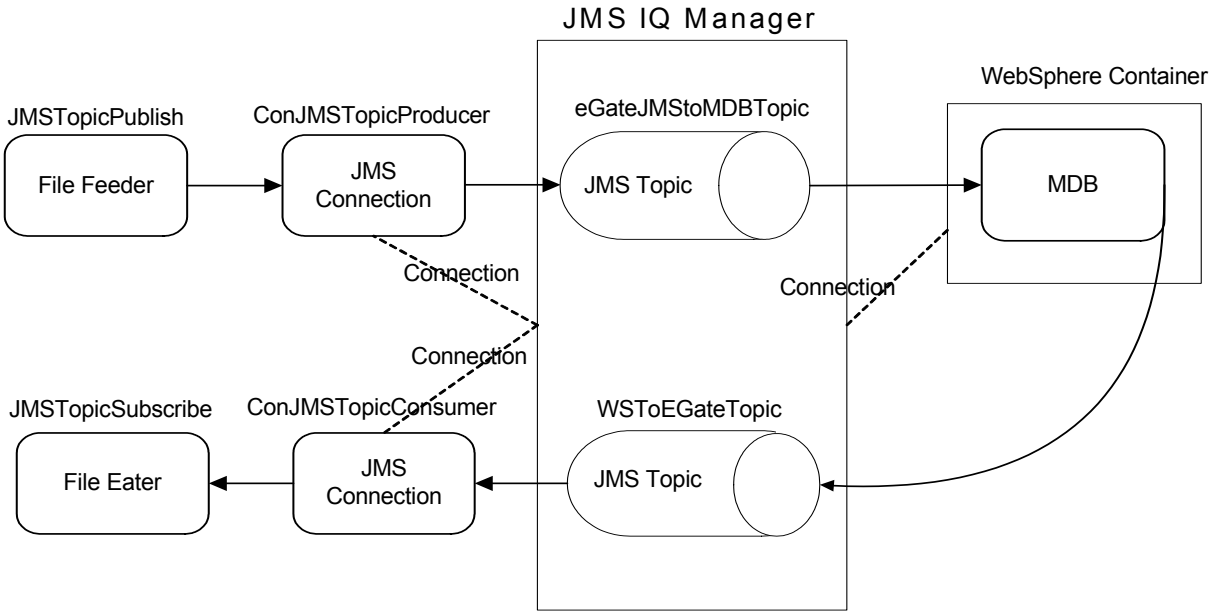
JMSTopicSubscriber e*Way

The JMSTopicSubscriber e*Way (stcewfile.exe) acts as an eater of messages coming from the WSToEGateTopic topic. The colJMSTopicSubscribe Collaboration subscribes to the conJMSTopicConsumer JMS Connection on the WSToEGateTopic topic. The conJMSTopicConsumer JMS Connection is configured to use the internal SeeBeyond JMS IQ Manager as the JMS server. The colJMSTopicSubscribe uses the crJMSTopicSubscribe Collaboration Rule, which displays the message received to standard output, and publishes the message to the external (writes the message received to a file).

JMSTopicPublish and JMSTopicSubscribe Message Flow

Figure 23 shows the components of the JMSTopicPublish and JMSTopicSubscribe sample.

Figure 23 JMSTopicPublish and JMSTopicSubscribe Sample Components



As shown in Figure 23, the File Feeder reads a file containing the input message event. A feeder Collaboration subscribes from external and publishes the input message, as a eGateJMSToMDBTopic event, to the JMS Connection. The JMS Connection is configured to use a JMS Topic, acting as a TopicPublisher. Both the JMS Connection and the MDB are configured to connect to the JMS IQ Manager as the JMS server. The MDB then publishes the message to WSToEGateTopic.

Configuring the JMSTopicPublish and JMSTopicSubscribe Sample

Once the sample has been successfully imported into e*Gate, you must configure it to correspond to the information as necessary.

Configuring the e*Ways

Each of the configuration files associated with the e*Way must be configured as needed, saved, and promoted to run time.

Configuration parameters for the JMSTopicPublish e*Way used with the JMSTopicPublish and JMSTopicSubscribe sample are the same as those in **Table 4 on page 57** with the exception of the following shown in Table 8.

Table 8 e*Way Configuration Parameters - JMSTopicPublish

e*Way Configuration Parameters	
General Settings - See Table 4 on page 57.	
Outbound (send) settings - See Table 4 on page 57.	
Poller (inbound) settings - Set as directed, otherwise see Table 4 on page 57.	
InputFileMask	*.tfin
Performance Testing - See Table 4 on page 57.	

Parameters for the JMSTopicSubscribe e*Way configuration used with the JMSTopicPublish and JMSTopicSubscribe sample appear as shown in Table 9.

Table 9 e*Way Configuration Parameters - ewJMSTopicSubscribe

e*Way Configuration Parameters	
General Settings - Set as directed, otherwise leave as default.	
AllowIncoming	NO
AllowOutgoing	YES
PerformanceTesting	NO
Outbound (send) settings - Set as directed, otherwise leave as default.	
OutputDirectory	C:\DATA
OutputFileName	topicrecv%d.dat
MultipleRecordsPerFile	NO
MaxRecordsPerFile	10000
AddEOL	Yes
Poller (inbound) settings - Set as directed, otherwise leave as default.	
PollDirectory	C:\INDATA
InputFileMask	*.fin
PollMilliseconds	1000
RemoveEOL	YES
MultipleRecordsPerFile	YES
MaxBytesPerLine	4096
BytesPerLineIsFixed	NO
File Records Per eGate Event	1
Performance Testing - Set as directed, otherwise leave as default.	
Performance Testing	100
InboundDuplicates	1

Configuring the e*Way Connection

Each of the configuration files associated with the e*Way must be configured as needed, saved, and promoted to run time.

The conJMSTopicProducer e*Way Connection Parameters associated with the JMSTopicPublish and JMSTopicSubscribe sample appear as shown in Table 10.

Table 10 e*Way Connection Parameters - conJMSTopicProducer

e*Way Connection Parameters	
General Settings - Set as directed, otherwise leave as default.	
Connection Type	Topic
Transaction Type	Internal
SDelivery Mode	Persistent
Maximum Number of Bytes to read	5000

Table 10 e*Way Connection Parameters - conJMSTopicProducer

e*Way Connection Parameters	
Default Outgoing Message Type	Text
Message Selector	
Factory Class Name	com.stc.commonService.SBYNJMSFactory
Message Service - Set as directed, otherwise leave as default.	
Server Name	localhost_iqmgr
Host Name	localhost
Port Number	26000
Maximum Message Cache Size	100

For more information on e*Way Connection Configuration Parameters for JMS see **Configuring the JMS e*Way Connection Parameters** on page 39.

The conJMSTopicConsumer e*Way Connection Parameters associated with the JMSTopicPublish and JMSTopicSubscribe sample appear as shown in Table 11.

Table 11 e*Way Connection Parameters - conJMSTopicConsumer

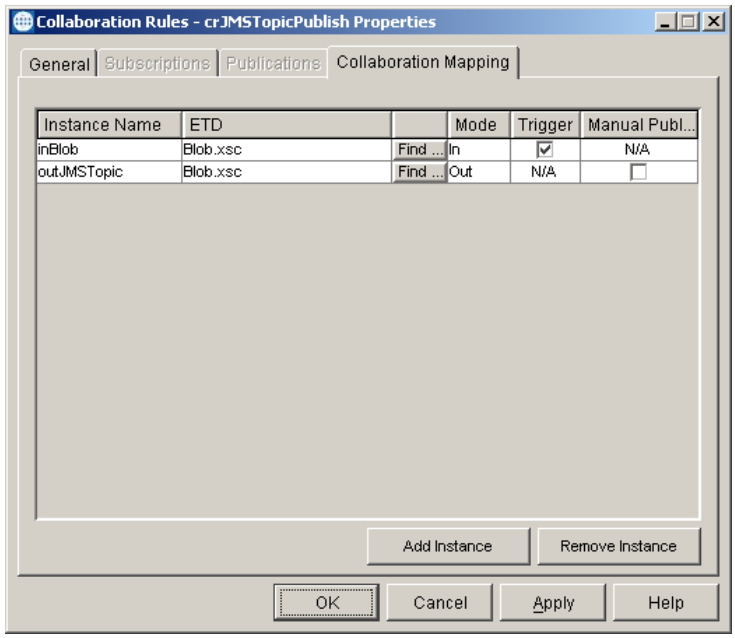
e*Way Connection Parameters	
General Settings - Set as directed, otherwise leave as default.	
Connection Type	Topic
Transaction Type	Internal
SDelivery Mode	Persistent
Maximum Number of Bytes to read	10000000
Default Outgoing Message Type	Text
Message Selector	
Factory Class Name	com.stc.commonService.SBYNJMSFactory
Message Service - Set as directed, otherwise leave as default.	
Server Name	localhost_iqmgr
Host Name	localhost
Port Number	26000
Maximum Message Cache Size	100

Creating Collaborations

JMSTopicPublish Collaboration Rule Mapping

The Collaboration Mapping associated with the crJMSTopicPublish Collaboration Rule appears as shown in Figure 24.

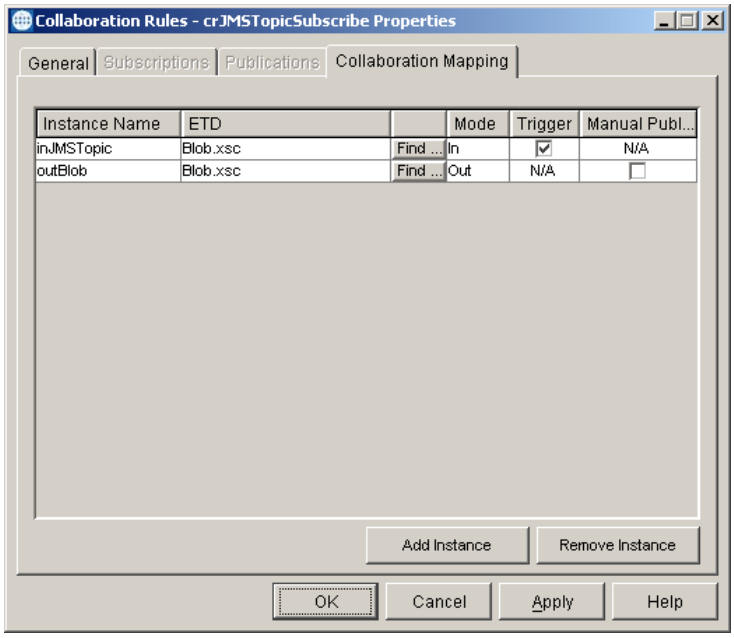
Figure 24 crJMSTopicPublish - Collaboration Mapping



JMSTopicSubscribe Collaboration Rule Mapping

The Collaboration Mapping associated with the crJMSTopicSubscribe Collaboration Rule appears as shown in Figure 25.

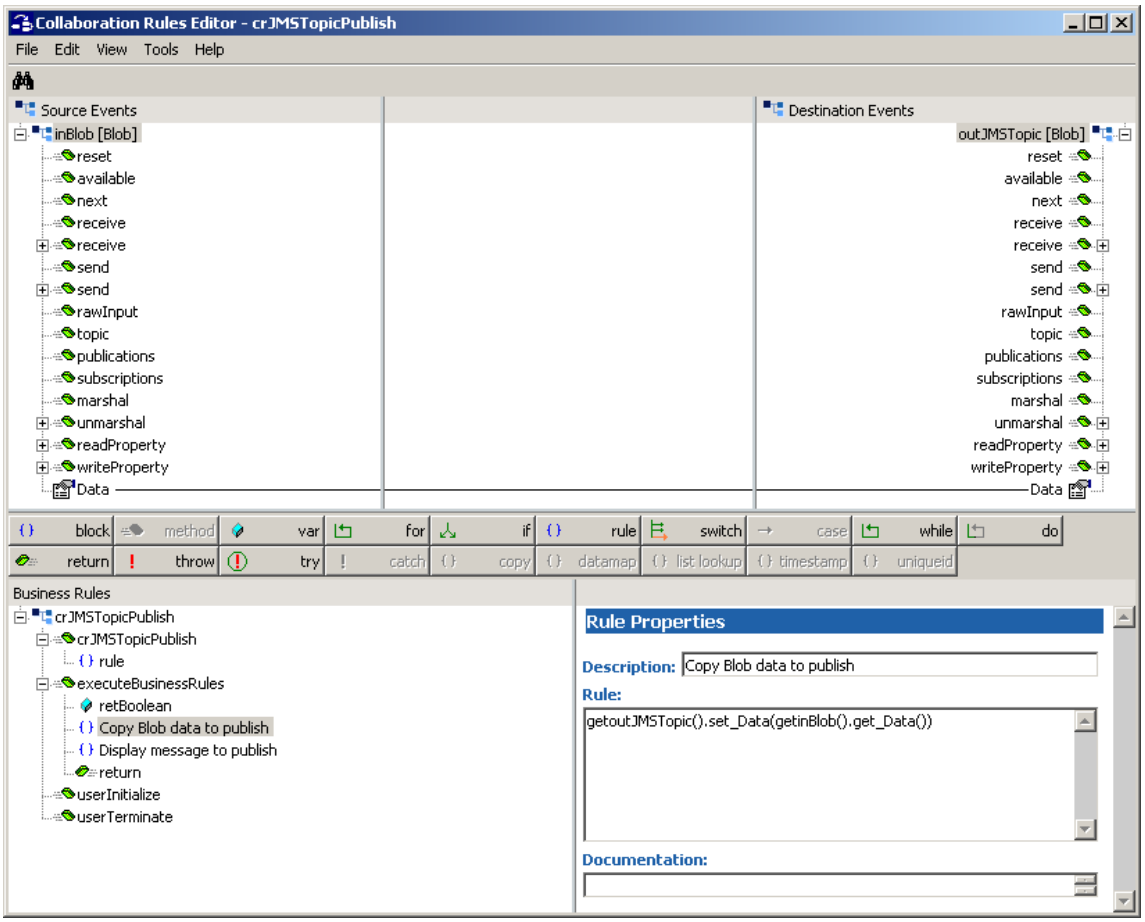
Figure 25 crJMSTopicSubscribe - Collaboration Mapping



crJMSTopicPublish Collaboration Rules

The crJMSTopicPublish Collaboration Rules appears as shown in Figure 26.

Figure 26 Collaboration Rules - crJMSTopicPublish



Each new rule is created by clicking the **rule** button on the Business Rules toolbar. For additional information on using the Java Collaboration Rules Editor, see the *e*Gate Integrator User's Guide*. The crJMSTopic Publish business rules are created as follows:

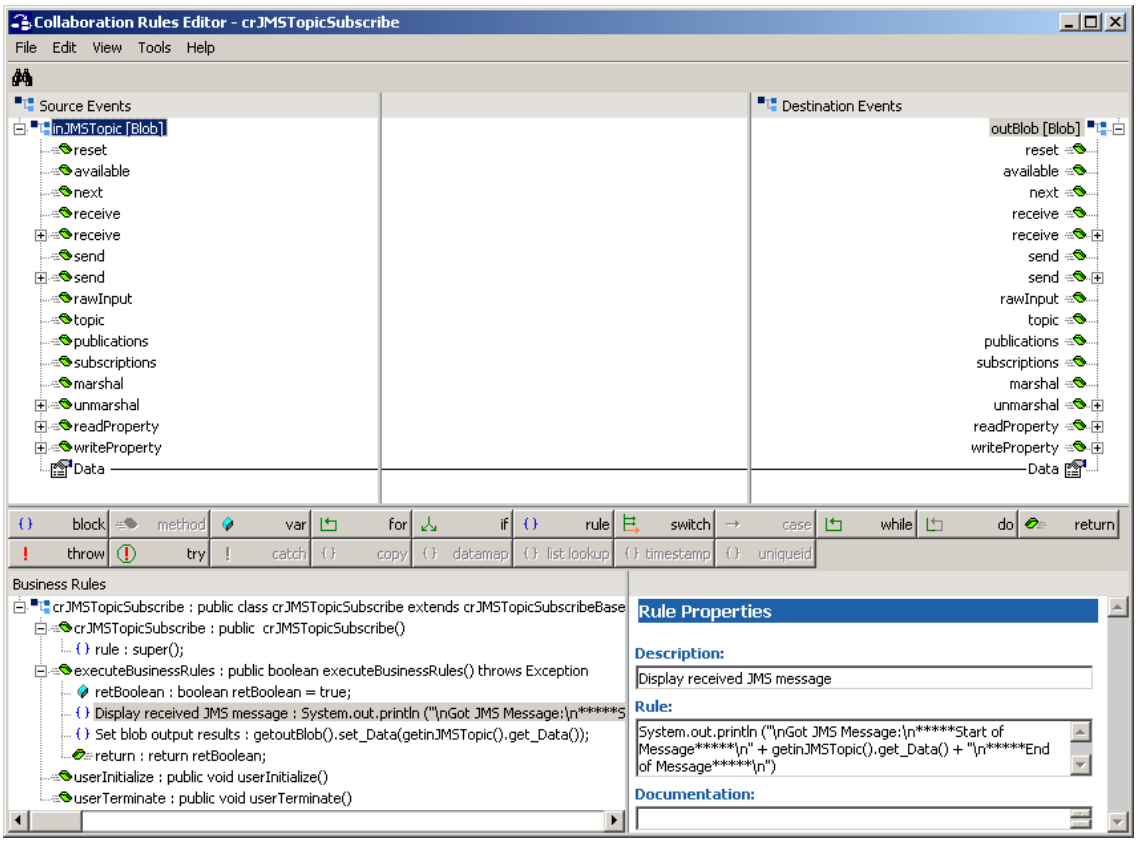
1. "**Copy blob data to Publish**" is created by dragging Data located under Source Events command node and dropping it on Data located under the Destination Events.
2. "**Display message to Publish**" is created by dragging **Data** located under Source Events command node into the Rule Properties, Rules window and entering code before and after to create the following code:

```
System.out.println("\nMessage to Publish:\n*****Start of Message*****\n" +
getinBlob().get_Data() + "\n*****End of Message*****\n")
```

JMSTopicSubscribe Collaboration Rules

The crJMSTopicSubscribe Collaboration Rules appear as shown in Figure 27.

Figure 27 Collaboration Rules - crJMSTopicSubscribe



Each new rule is created by clicking the **rule** button on the Business Rules toolbar. For additional information on using the Java Collaboration Rules Editor, see the *e*Gate Integrator User's Guide*.

The crJMSTopicSubscribe business rules are created as follows:

1. "**Display received JMS message"** is created by dragging **Data** located under Source Events command node into the Rule Properties, Rules window and entering code before and after to create the following code:

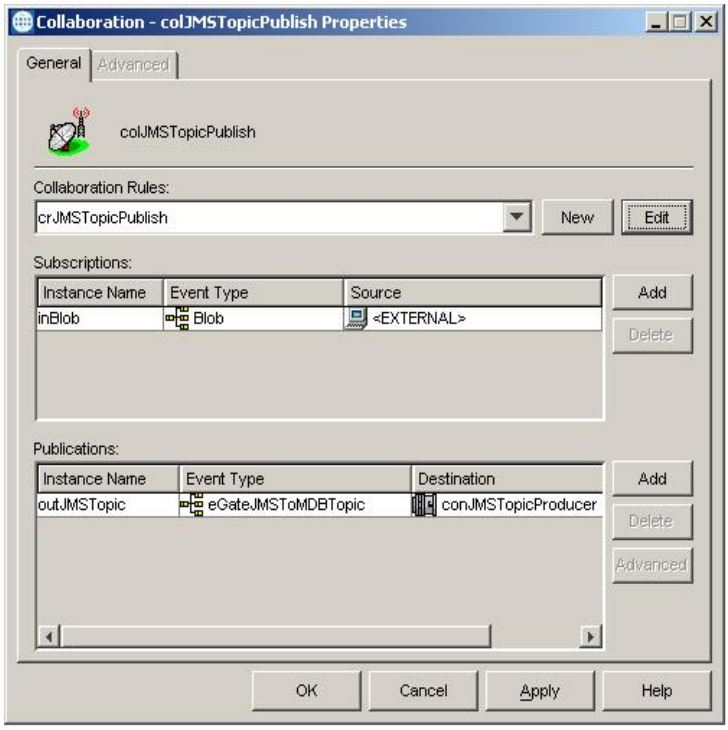
```
System.out.println ("\nGot JMS Message:\n*****Start of
Message*****\n" + getinJMSTopic().get_Data() + "\n*****End of
Message*****\n")
```

2. "**Set blob output results**" is created by dragging Data located under Source Events command node and dropping it on Data located under the Destination Events.

JMSTopicPublish Collaboration Properties

The colJMSTopicPublish Collaboration properties for the JMSQueueSend sample appear as shown in Figure 28.

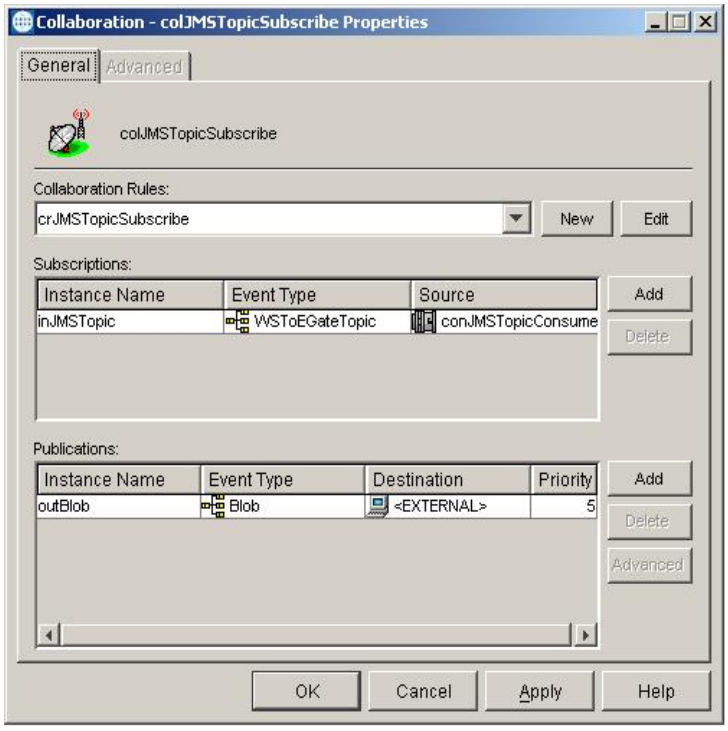
Figure 28 colJMSTopicPublish - Collaboration Properties



JMSTopicSubscribe Collaboration Properties

The colJMSTopicSubscribe Collaboration properties for the JMSTopicSubscribe sample appear as shown in Figure 29.

Figure 29 colJMSTopicSubscribe - Collaboration Properties



4.4.5. JMSXATopicPublish Sample

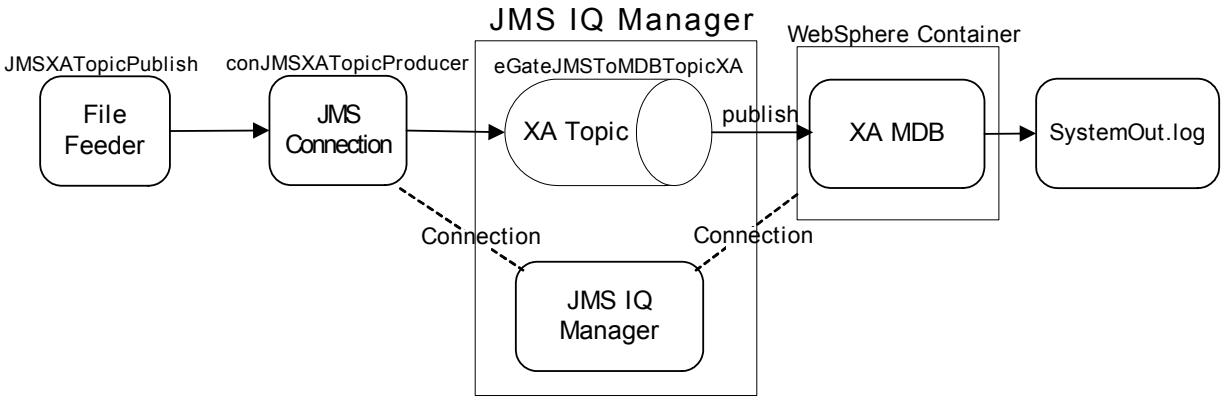
In this sample, the JMSXATopicPublish e*Way (stcewfile.exe) acts as a feeder of messages to the eGateJMSToMDBTopicXA topic. The JMSXATopicPublish e*Way looks for files with the extension "*.xatfin" as input files (the input directory configured is C:\INDATA). The colJMSXATopicPublish Collaboration subscribes to external for an event from a file and publishes to the conJMSXATopicProducer JMS Connection. The conJMSXATopicProducer JMS Connection is configured to use the internal SeeBeyondJMS IQ Manager as the JMS server.

The colJMSXATopicPublish Collaboration uses the crJMSTopicXAPublish Collaboration rule which copies data from the source event and writes to eGateJMSToMDBTopicXA. The MDB receives the message and prints to the **SystemOut.log** file in **WebSphere/logs/<servername>**.

JMSXATopicPublish Message Flow

Figure 30 shows the components of the JMSXATopicPublish sample.

Figure 30 JMSXATopicPublish Sample Components



Configuring the JMSXATopicPublish Sample

Once the sample has been successfully imported into e*Gate, you must configure it to correspond to the information as necessary.

Configuring the e*Ways

Each of the configuration files associated with the e*Way must be configured as needed, saved, and promoted to run time.

Configuration Parameters for **ewJMSXATopicPublish** e*Way configuration used with JMSXATopicPublish are the same as those in **Table 10 on page 68**.

Configuring the e*Way Connection

Each of the configuration files associated with the e*Way must be configured as needed, saved, and promoted to run time.

Connection Parameters for conJMSXATopicProducer e*Way connection must be set as shown in Table 12.

Table 12 e*Way Connection Parameters - conJMSXATopicProducer

e*Way Connection Parameters	
General Settings - Set as directed, otherwise leave as default.	
Connection Type	Topic
Transaction Type	XA-Compliant
SDelivery Mode	Persistent
Maximum Number of Bytes to read	10000000
Default Outgoing Message Type	Text
Message Selector	
Factory Class Name	com.stc.commonService.SBYNJMSFactory
Message Service - Set as directed, otherwise leave as default.	
Server Name	localhost_iqmgr
Host Name	localhost

Table 12 e*Way Connection Parameters - conJMSXATopicProducer

e*Way Connection Parameters	
Port Number	26000
Maximum Message Cache Size	100

For more information on e*Way Connection Configuration Parameters for JMS see **Configuring the JMS e*Way Connection Parameters** on page 39.

Creating Collaborations

JMSXATopicPublish Collaboration Rule Mapping

The JMSXATopicPublish sample uses the crJMSTopicXAPublish Collaboration Mapping (see **Table 25 on page 70**).

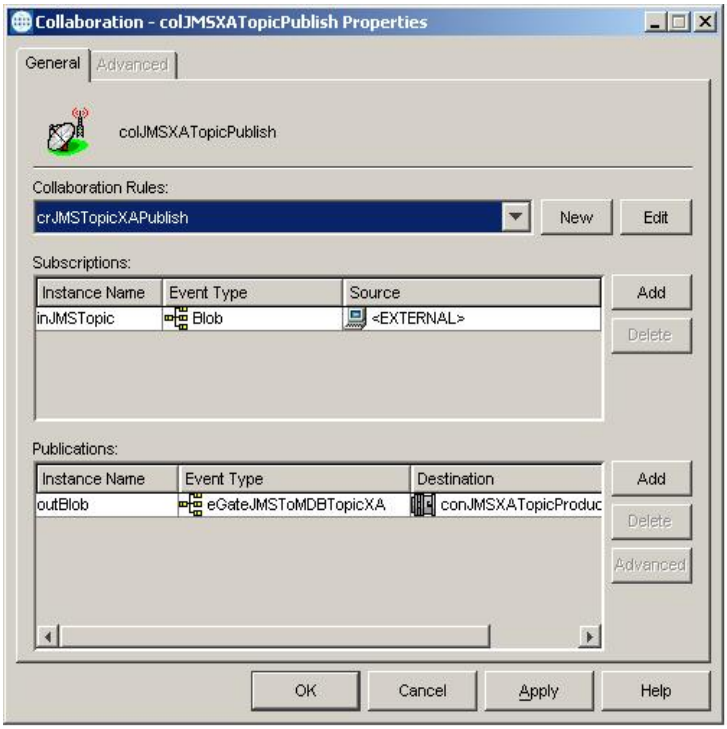
JMSXATopicPublish Collaboration Rules

The JMSXATopicPublish Sample uses the crJMSTopicXAPublish Collaboration Rule (see **Table 27 on page 72**).

JMSXATopicPublish Collaboration Properties

The colJMSTopicPublish Collaboration for the JMSXATopicPublish sample appears as shown in Figure 31.

Figure 31 colJMSXATopicPublish - Collaboration Properties



4.5 Executing the Sample Schemas

Execute the WebSphere sample schemas as follows:

1. At the command prompt, enter the following:

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

Note: *Substitute the italicized values with the specific values for your schema.*

2. Start the Schema Manager.
3. When prompted, enter the host name which contains the Registry Host started in step 1 above.
4. Select the sample schema.
5. Verify that the Control Broker is connected (the message on the Control tab of the console will indicate command succeeded and the status as up).
6. Right-click the IQ Manager (**hostname_iqmgr**) and click **Start**.
7. Right-click each e*Way and click **Start**.
8. View the output by copying the output file (specified in the Outbound e*Way configuration file) to another location. Open the file in the new location.

Note: *Do not open the destination file while the schema is running. This will cause errors.*

Index