SeeBeyond ICAN Suite

eGate API Kit Developer's Guide

Release 5.0.5

Version 20041111092601.

Contents

Chapter 4

Implementing JMS in Java or COM+ 28

Chapter 5

Client Libraries for the SeeBeyond COM+ APIs 76

Chapter 8

Chapter 9

Chapter 10

Chapter 11

Chapter 1

Introduction

The eGate API Kit enables you to create custom applications or modify existing external applications to interface with the eGate system. The API Kit provides the following interfaces:

Message Service Interfaces

- Java
- COM +
- C/C++

1.1 Overview

The eGate API Kit provides a delivery service mechanism using the Message Service. This Message Service is defined by Sun's JMS version 1.0.2 and is available as a Javadoc. It can be downloaded from:

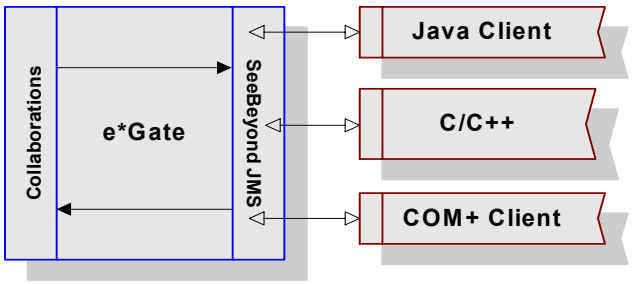
http://java.sun.com/products/jms/docs.html

1.1.1. Message Service Functionality

The Message Service provides applications with an API set for a common and elegant programming model that is portable across messaging systems. Enterprise messaging systems are used to send notification of events and data between software applications.

The basic Message Service data flow is illustrated in the following figure:

Figure 1 Basic Message Service Data Flow



Several common programming models are supported by the Message Service API, including the following:

- **Publish/Subscribe (pub/sub)**
- **Point-to-point (P2P)**
- **Request/Reply**
- **Message selector**

Each of these programming models are briefly described below.

Publish-and-Subscribe (Pub/Sub)

In a publish-and-subscribe scenario, one producer can send a single message to multiple consumers via a virtual channel called a *topic*. Consumers must *subscribe* to a topic to be able to receive it. Any messages addressed to a specific topic are delivered to all of that topic's consumers (*subscribers*).

The pub/sub model is predominantly a push-based model, in that messages are automatically broadcast to consumers without the consumers having to request or poll the topic for new messages.

Point-to-Point (P2P)

In point-to-point messaging systems, messages are routed to an individual consumer which maintains a *queue* of incoming messages. Messaging applications *send* messages to a specified queue, and clients *retrieve* messages from a queue. In a point-to-point scenarios, each message is delivered to exactly one client. JMS uses the term Queue for P2P MessageQueues.

Request/Reply

When the client sends a message and expects to receive a message in return, request/reply messaging (a synchronous object-messaging format) can be used. Request/reply uses either pub/sub or point-to-point to enable the functionality.

JMS does not explicitly support request/reply messaging, although it allows it in the context of the other methods.

Message Selector

Many messaging applications require the additional functionality of filtering and categorization of the messages they produce. If a message is sent to a single receiver, this can be done by including the criteria in the message. The receiving client can then discard the messages not required.

However, when a message must be distributed to many clients, the JMS provider can handle much of the filtering and routing (without impacting each client application) if the criteria is included in the message header.

Clients include application-specific selection criteria in messages via the message properties. Clients specify message selection criteria via JMS message selector expressions.

Additional Supported Resources

The SeeBeyond Message Service supports external resources that you can use when developing your messaging systems, including:

- **Java Naming and Directory Interface (JNDI)**
- **Compensating Resource Manager**

These resources are described below.

Java Naming and Directory Interface (JNDI)

The Java Naming and Directory Interface (JNDI) provides naming and directory functionality to applications written using Java. JNDI consists of the following:

- **An API set**
- **A service provider interface (SPI)**

Java applications use the JNDI API to access naming and directory services. The SPI allows the naming and directory services to be accessed transparently, thus providing the JNDI API access to their services.

JNDI is included in the Java 2 SDK, v 1.3 and later releases.

To use the JNDI functionality, the JNDI classes are required, along with one or more service providers (such as, LDAP, CORBA, or RMI).

Compensating Resource Manager

The Compensating Resource Manager (CRM) provides support for distributed transaction with multiple resource managers. These COM+ objects perform non-database operations as part of a distributed transaction. A distributed transaction involves multiple independent resource managers. If any part of the transactions fail, the whole transaction fails.

Important: *CRM is only supported on Windows 2000.*

1.2 Supported Operating Systems

The Message Service APIs are available on the following operating systems:

- Windows 2000, Windows XP, Windows Server 2003
- HP NonStop Server G06.22
- HP Tru64 V5.1A
- HP-UX 11.0, 11i (PA-RISC), and 11i v2.0 (11.23)
- IBM AIX 5.1L and 5.2

- Red Hat Enterprise Linux AS 2.1 (Intel x86)
- Red Hat Linux 8 (Intel x86)
- Sun Solaris 8 and 9
- SuSE Linux Enterprise Server 8 (Intel x86)
- Japanese Windows 2000, Windows XP, Windows Server 2003
- Japanese HP-UX 11.0, 11i (PA-RISC), and 11i v2.0 (11.23)
- Japanese IBM AIX 5.1L and 5.2
- Japanese Sun Solaris 8 and 9
- Korean Windows 2000, Windows XP, Windows Server 2003
- Korean HP-UX 11.0, 11i (PA-RISC), and 11i v2.0 (11.23)
- Korean IBM AIX 5.1L and 5.2
- Korean Sun Solaris 8 and 9

Note: *Double-byte character sets (DBCS) are not supported for the C API on Solaris and HP-UX operating systems.*

1.3 System Requirements

1.3.1. Java Message Service Java API Requirements

To use the Message Service APIs, you need the following:

- A TCP/IP network connection.
- JDK 1.4.

1.3.2. Java Message Service COM+ API Requirements

To use the Message Service COM+ APIs, you need the following:

- A TCP/IP network connection.
- A development environment with a compiler that is compatible with operating systems supported by eGate; for example, Microsoft Visual .Net 2003.

1.3.3. Java Message Service C/C++ API Requirements

To use the Message Service for C/C++ on UNIX operating systems with this kit, you will need GNUMake. All of the samples were created using this program. See the Makefile .args for suggested UNIX compilers.

When creating executables, you will need to select option **-k** within GNUMake to assure that if any errors are encountered, the error will be skipped and the program will continue.

- A TCP/IP network connection.
- C/C++: A development environment with a compiler that is compatible with operating systems supported by eGate; for example, Microsoft Visual .Net 2003.

1.4 Compilers

Before you can use the samples that are presented with the kit, you will need a compiler. The following compilers are recommended. If you choose to use a different compiler than those listed, be aware that it is possible that some compilers will not be compatible with the eGate environment.

Table 1 Recommended Compilers

Operating System	Compiler
Windows	.Net 2003 version 7.1
AIX	Visual Age for C++ 6.0, 64 bit
AIX	Visual Age for C++ 6.0, 32bit
Solaris	Sun ONE Studio 7
HP - UX	aCC 3.37, 64 bit
HP Itanium	aCC 5.36, 64 bit
Linux 8 (Red Hat 8	GCC V3.2-7
Linux Server 2.1 (Red Hat Enterprise Linux 2.1)	Advanced Server 2.1
Tru64	Tru64 5.1A

Chapter 2

Installing the eGate API Kit

This chapter describes the process of installing the eGate API Kit.

This chapter contains

2.1 Installing the eGate API Kit on Windows or UNIX

The installation process includes:

- Installing the ICAN Repository.
- Uploading products to the Repository (including the eGate API Kit, documentation, and sample files).
- Downloading components (including Enterprise Designer, the Logical Host, and the eGate API Kit) from the Repository.
- Updating products in the Enterprise Designer using the Update Center Wizard.

To install the eGate API Kit

Follow the instructions for installing ICAN in the *SeeBeyond ICAN Suite Installation Guide*, and include the following steps:

1. After uploading **eGate.sar** to the Repository, upload the following additional product files:
   - **eGate_APIKit<operating_system>.sar** (for example, **eGate_APIKit_SunOS.sar**)
   - **eGateAPIKitDocs.sar**

Note: *These files may not be located on the same installation disc as the* ***eGate.sar*** *file.*

2. From the Downloads tab of the Enterprise Manager, click the link for the version of the eGate API Kit you wish to install; for example, **API kit for Windows 2000, 2003, and XP**. (There is a separate link for each version of the API Kit you uploaded in the previous step).
3. Use WinZip to extract the API Kit files to the desired location.

4 Repeat steps 1 through 3 for each platform you wish to install.

If you install the API Kit for multiple operating systems, take care on extract the files for each operating system to a unique location.

To install the eGate API Kit samples

1 From the Documentation tab of the Enterprise Manager, click **eGate API Kit** to view the list of files available for this product.

2 Click **Download Sample** to open the **eGate_API_Kit_Sample.zip** file.

3 Use WinZip to extract the sample files to the desired location.

4 After you complete the process of installing the Repository, Logical Host, and Enterprise Designer (as described in the *SeeBeyond ICAN Suite Installation Guide*), refer to the *eGate Integrator User's Guide* for instructions on installing the sample project into your repository via the Enterprise Designer.

2.2 Installing the eGate API Kit on HP NonStop Server

This section assumes you have already completed the instructions in the *SeeBeyond ICAN Suite Installation Guide* to install your ICAN Repository, Logical Host(s), and the Enterprise Designer on your client workstations.

The eGate API Kit product files are installed on your Repository as part of the HP NonStop Server Repository installation process. The only item remaining to be installed is the sample Project.

To install the HP NonStop JMS sample Project

1 In the Documentation tab of the Enterprise Manager, click **eGate API Kit** to view the list of files available for this product.

2 Click **Download Sample** to open the **eGate_API_Kit_Sample.zip** file.

3 Use WinZip to extract the **NSJMSProj.zip** file to the desired location.

4 Refer to the *eGate Integrator User's Guide* for instructions on installing the **NSJMSProj** Project.

For more information

For more information on using the eGate API Kit with HP NonStop Server, refer to **Chapter 11 "HP NonStop APIs" on page 388**.

Chapter 3

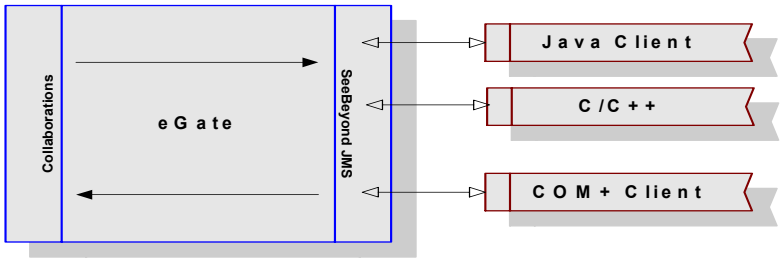
Configuring the Message Service

This chapter explains how to configure the three separate components that constitute SeeBeyond Message Service:

- Message Service Client: the external application
- Message Service: the data container and router
- Kit Connection: the link between eGate and the external system

The following diagram illustrates the communication between each component.

Figure 2 Message Service Communication Architecture

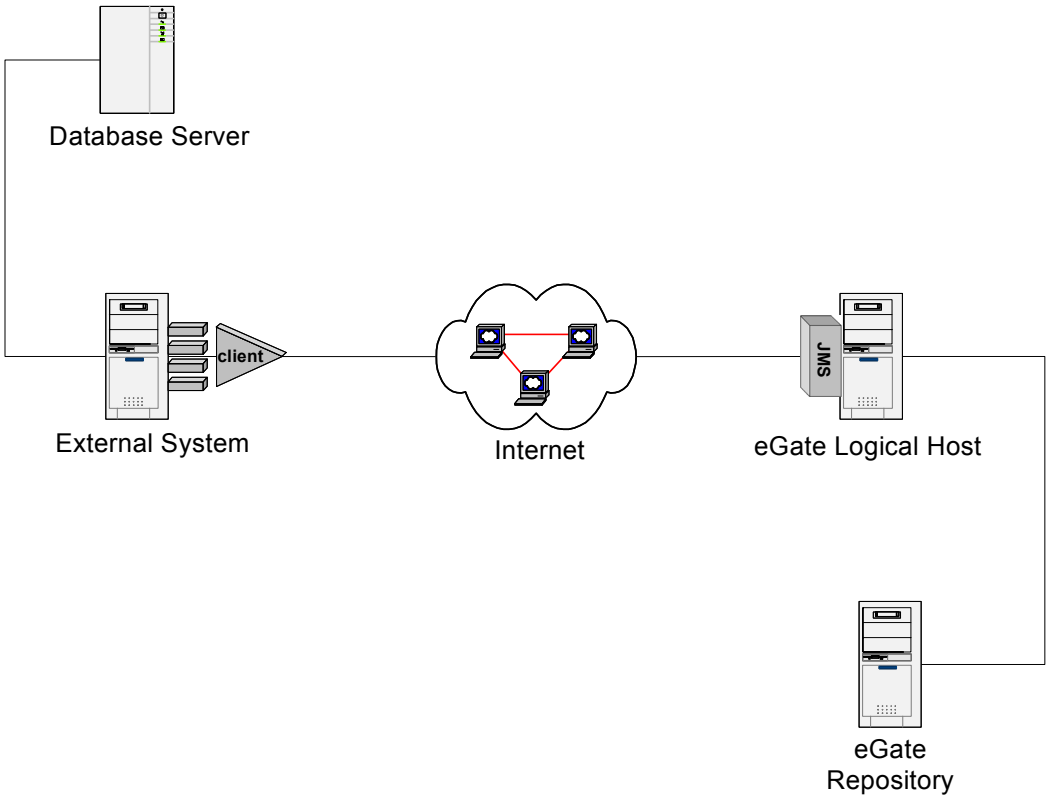


3.1 Message Service Clients

The current Message Service supports Java, COM+, C, and C++ clients. The sections that follow provide the information necessary to configure these clients.

In the diagram that follows all of the necessary components have been isolated onto a separate machine. While this separation is not mandatory, the combinations of components that reside together on various machines, change depending upon your needs.

Figure 3 TCP/IP Communication Architecture



In some form, the following components must exist:

- eGate Repository Host (eGate Server)
- eGate Logical Host (eGate Client)
- External System (Message Service Client file)
- Database Server (Data Repository)

Important: *From this point forward, when referring to a machine, the above naming conventions are used. Remember that multiple components may reside on the same machine. For example, the eGate Logical Host and the External System may exist on one physical machine.*

For information about the architecture and specific operation see the *eGate Integrator JMS Reference Guide.*

3.1.1. Considerations

When writing any external JMS code you must know the expected data format (byte or text), the name of the Topic/Queue, the name of host and port number of the JMS client. For example, there is no way to dynamically determine the port number of the JMS IQ Manager, so you must ascertain this in advance.

Segment size (512 bytes/page for Windows, 1024 bytes for UNIX) must always be larger than the largest expected Event, preferably by an order of magnitude.

For more information about JMS IQ Managers, see the *eGate Integrator JMS Reference Guide*.

3.1.2. Setting up the Java Client

Once the eGate API Kit has been successfully installed, additional steps are required to run Java JMS client programs. In this section, the setup steps are included for setting up the Message Service to use Java.

To begin using the Message Service, do the following:

Locate the **com.stc.jms.jmsis.jar, jta.jar, jms.jar,** and **log4j.jar** files in the directory on the external machine that you have installed the eGate API Kit on.

Modify the **CLASSPATH** on your external system to include the **jms.jar**, **com.stc.jmsis.jar**, and **log4j.jar** (when connecting to JNDI through the Information Server) files. For XA support, you will also need to include the **jta.jar** in your path.

If you are connecting to JNDI on the Integration Server, also include the **com.stc.jmsis.jar** in your classpath.

When using the STCMS client library on HP-UX operating systems, make sure that the JVM has been started with the `-XdoCloseWithReadPending` flag. This can be done by setting this flag on the command line; e.g. to run sample X, the command line should read:

```
java -XdoCloseWithReadPending -cp com.stc.jmsis.jar;.;jms.jar Sample
```

3.1.3. Setting up the COM+ Client

Once the eGate API Kit has been installed successfully, additional steps are required to finish the setup, before data exchange can begin. In this section, the setup steps are included for setting up the Message Service to use COM+.

For all COM+ implementations of the Message Service for COM+, do the following:.

Locate the **stc_mscom.dll**, **stc_msclient.dll**, **stc_mscommon.dll**, **stc_msapi.dll** files in the directory on the external system that you have installed the eGate API Kit on. From the command prompt of the external system, register the file **stc_mscom.dll** into the Windows Registry by doing the following:

```
regsvr32 <root>\apikit\jms\complus_api\stc_mscom.dll
```

Viewing the Message Service COM+ APIs

You can view the Message Service COM+ APIs using any application that is capable of viewing COM+ APIs. We suggest Microsoft Visual Basic .Net 2003 as the viewing application.

Compensating Resource Manager (CRM)

A *Compensating Resource Manager* can be described as a COM+ object that uses a set of tools (CRM facility) enabling the user to create resource managers. This allows the user to perform non-database operations (such as generating a file) as part of a transaction.

A *distributed transaction* is a transaction that involves multiple independent resource managers. For example, it might include an Oracle database at the corporate office and a SQL Server database at the partner's warehouse. The involved resource managers attempt to complete and commit their part of the transaction. If any part of the transaction fails, all resource managers roll back their respective updates.

This is accomplished using the two-phase commit protocol. In this protocol, the activity of one or more resource managers is controlled by a separate piece of software called a transaction coordinator.

CRM Architecture

A minimum of two COM components must be implemented to create a CRM scenario. At least one CRM Worker, and a CRM Compensator are required. The COM+ CRM functionality provides the CRM clerk and a durable log file. The CRM Worker contains the application-level code that directs the business logic employed by the Compensating Resource Manager. If the CRM writes XML files, the CRM Worker is likely to contain a WriteToFile method, along with a COM+ implementation of JMS interfaces to the message service. The CRM Worker acts as a transacted COM+ component that is configured to require a transaction. When an application activates a CRM Worker component, the CRM Worker instantiates the CRM clerk object, and uses that CRM clerk to register a compensator component.

The functionality provided by SeeBeyond's implementation of CRM is contained within the COM+ library, **stc_mscom.dll**.

The CRM Worker is implemented via the following classes:

- XAConnection
- XAConnectionFactory
- XAQueueConnection
- XAQueueConnectionFactory
- XAQueueSession
- XARecord
- XASession
- XATopicConnection
- XATopicConnectionFactory
- XATopicSession

The CRM Compensator is implemented in the Compensator file.

When the transaction in which the CRM Worker is participating commits, the DTC calls methods contained within the CRM Compensator interface that the CRM Compensator must implement. The DTC makes these calls at each step of a two-phase commit protocol. If the prepare phase is successful, the updates are made permanent by committing the changes. If any part of the complete transaction fails, the transaction rolls back the information, aborting the transaction.

Two-phase Commit Protocol

Two-phase commit is the key to implementing distributed transactions. The activity of one or more resource managers is controlled by the transaction coordinator. There are five steps in the two-phase commit protocol.

1. An application invokes the commit method in the transaction coordinator.
2. The transaction coordinator contacts the various resource managers relevant to the transaction, and directs them to prepare to commit the transaction. (Begin phase one.)
3. The resource manager must be able to guarantee the ability to commit the transaction, or perform a rollback. Most resource managers write a journal file, containing the intended changes to durable storage. If unable to prepare the transaction, a negative response is set to the transaction coordinator.
4. All responses from the involved resource managers are collected.
5. The transaction coordinator informs the involved resource managers. (Phase Two) If any of resource managers responded negatively, the transaction coordinator sends a rollback command. If all of the resource managers responded affirmatively, the transaction coordinator directs all of the resource managers to commit the transaction. The transaction cannot fail after this point.

Compensating Resource Manager (CRM) Setup

To enable SeeBeyond's CRM functionality, the following steps are required.

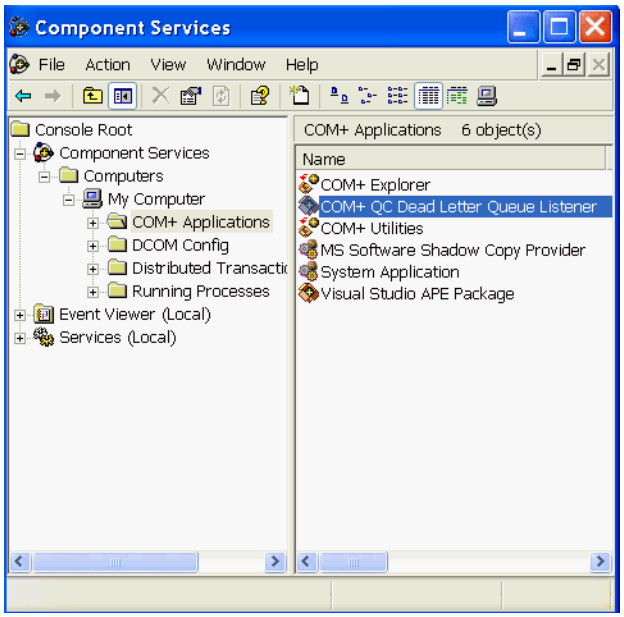
1. From a command prompt of the external system, register the file **stc_mscom.dll** into the Windows Registry by entering the following command

```
regsvr32 your_path_location\stc_mscom.dll
```

2 Open the Component Services applet (**Start -> Settings->Control Panel -> Administrative Tools -> Component Services**).

3 Expand the **Component Services** folder (see Figure 4) and right-click **COM+ Applications**.

Figure 4 Component Services Folder

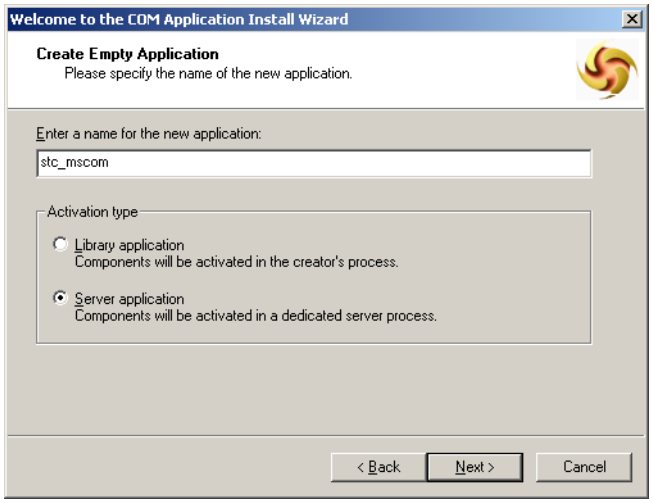


4 On the shortcut menu, click **New\Application**. The COM Application Install Wizard opens. Click **Next** to continue.

5 In the **Install or Create** step, click **Create an empty application**.

6 In the **Create Empty Application** step, enter the name **stc_mscom** and click the option button **Server application** (as in Figure 5), and then click **Next**.

Figure 5 COM Application Install Wizard: New Application

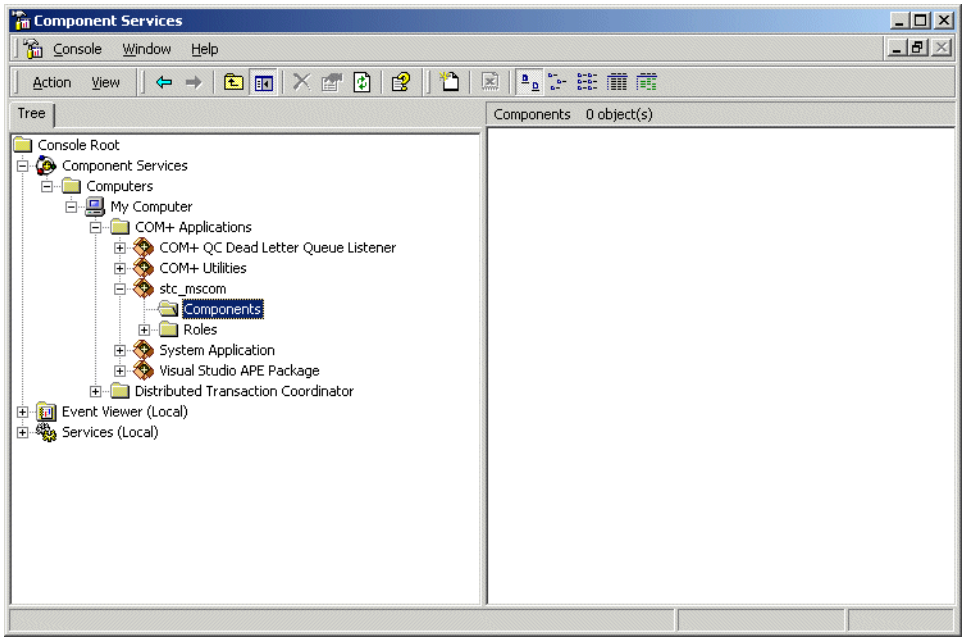


7 In the **Set Application Identity** step, click **Interactive User**, and then click **Next**.

8 Click **Finish**.

9 Expand the **stc_mscom** component; see Figure 6.

Figure 6 Component Services: stc_mscom Component

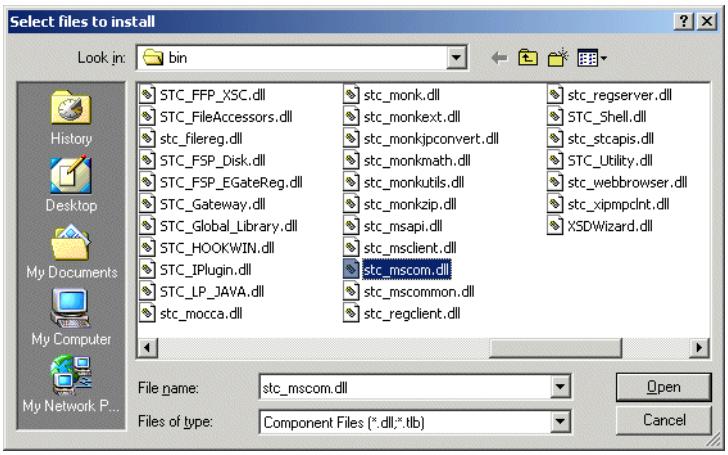


10 Right-click the **Components** folder. On the shortcut menu, click **New Component**.

11 From the COM Component Install Wizard. Click **Install new component(s)**.

12 In the **Install new components** step, click **Add** to open the **Select Files to Install** dialog box (see Figure 7). Locate and open the file **stc_mscom.dll**.

Figure 7 COM Component Install Wizard

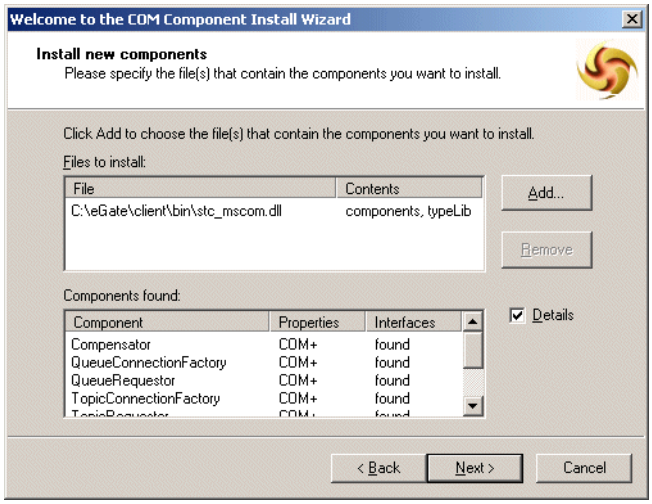


- If you are running the **.dll** on the same machine where eGate was installed, the file is located in *<eGate>***client\bin** directory.

- If **stc_mscom.dll** has been copied to another system, the file is located in the directory where you pasted it previously.

13 Once the component appears in the **Components found** pane, ensure that the **Details** box is selected (as in Figure 8), and then click **Next** to continue.

Figure 8 COM Component Install Wizard: Add

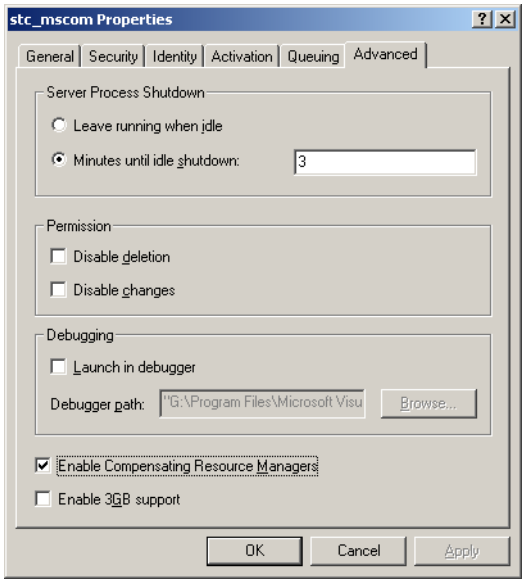


14 Click **Finish**.

15 Right-click the **stc_mscom** component and, on the shortcut menu, click **Properties**.

16 The **stc_mscom Properties** dialog box appears. In the Advanced tab, ensure **Enable Compensating Resource Manager** is selected (see Figure 9), and then click **OK**.

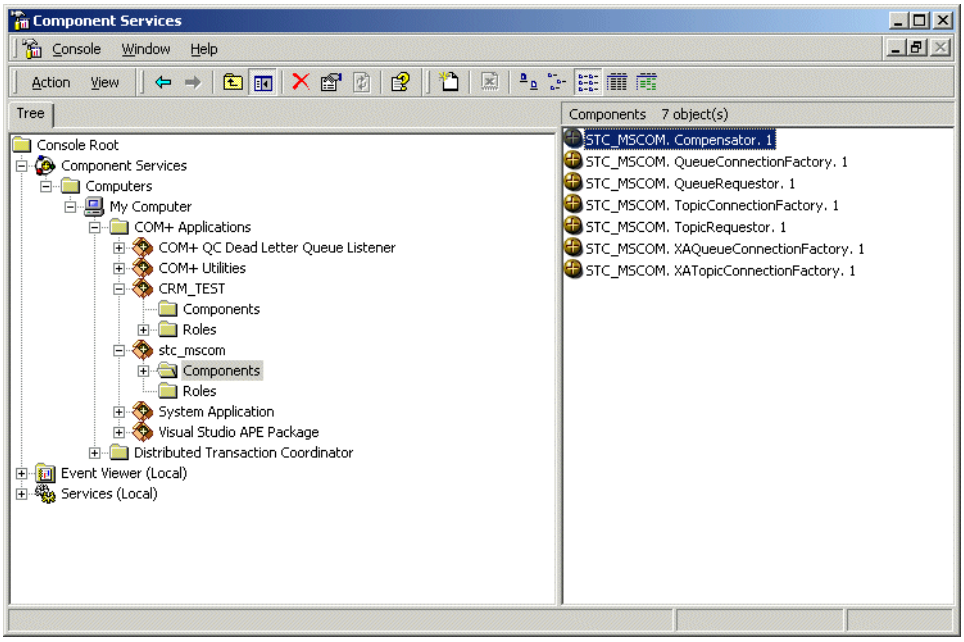
Figure 9 stc_mscom Properties: Advanced



17 Expand the **stc_mscom** component and click the **Components** folder to view the objects it contains.

18 In the **Components** pane, on the right side of the window; see Figure 10, right-click **STC_MSCOM.Compensator**, and, on the shortcut menu, click **Properties**.

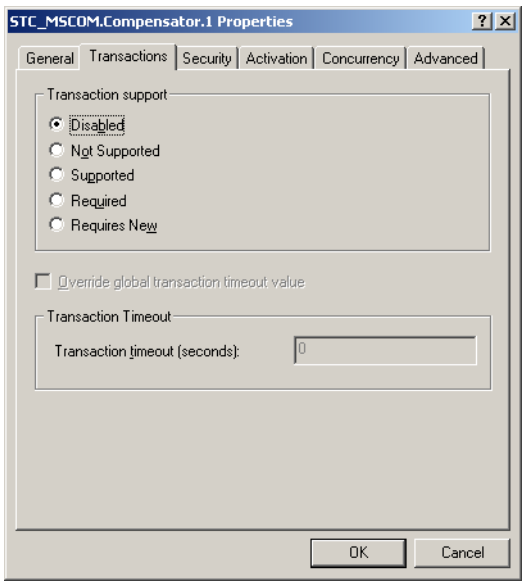
Figure 10 STC_MSCOM.Compensator Properties



The **STC_MSCOM.Compensator Properties** dialog box appears.

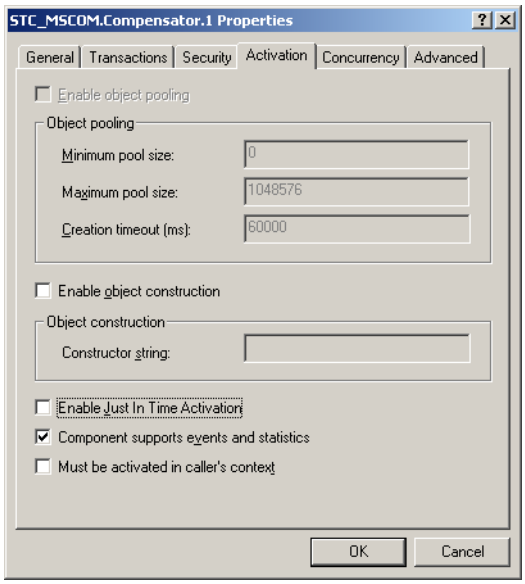
19 In the Transactions tab: For **Transaction support**, click **Disabled** (see Figure 11).

Figure 11 STC_MSCOM.Compensator Properties:Transaction Support



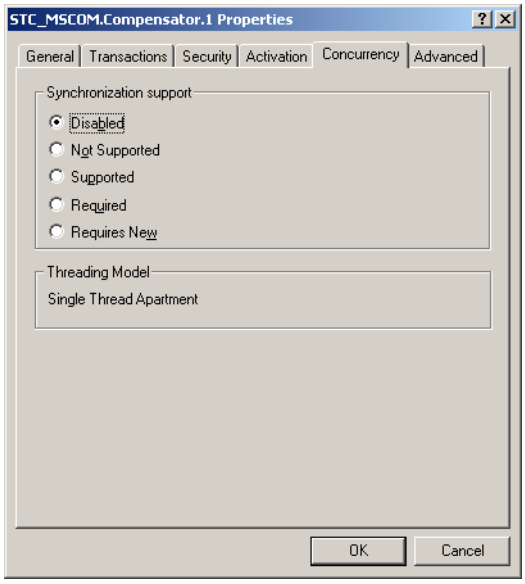
20 In the Activation tab: Clear **Enable Just In Time Activation** (see Figure 12).

Figure 12 STC_MSCOM.Compenstator Properties:Activation



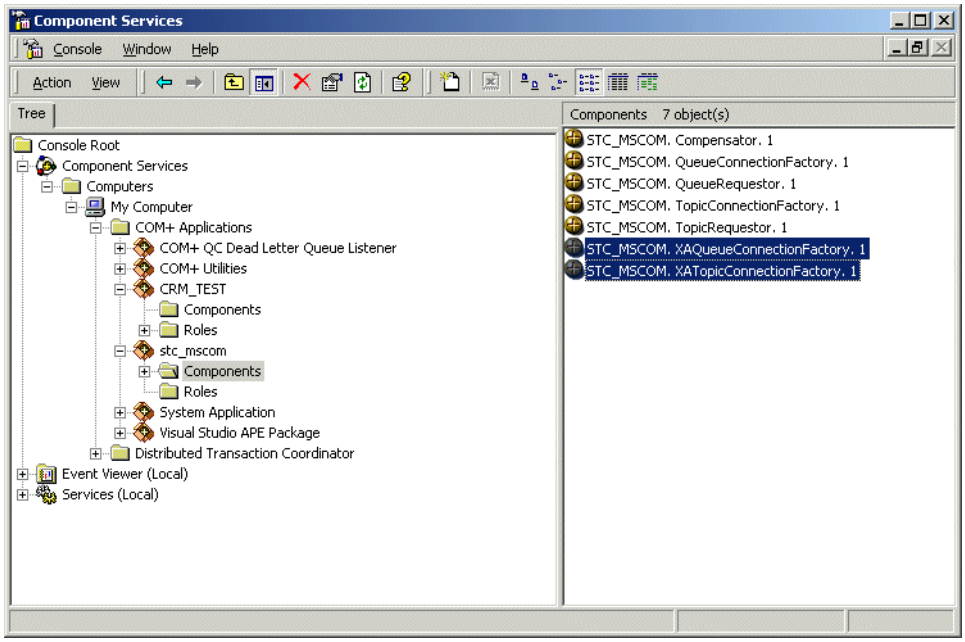
21 In the Concurrency tab: For **Synchronization support**, click **Disabled** (see Figure 13), and then click **OK**.

Figure 13 STC_MSCOM.Compensator Properties:Concurrency



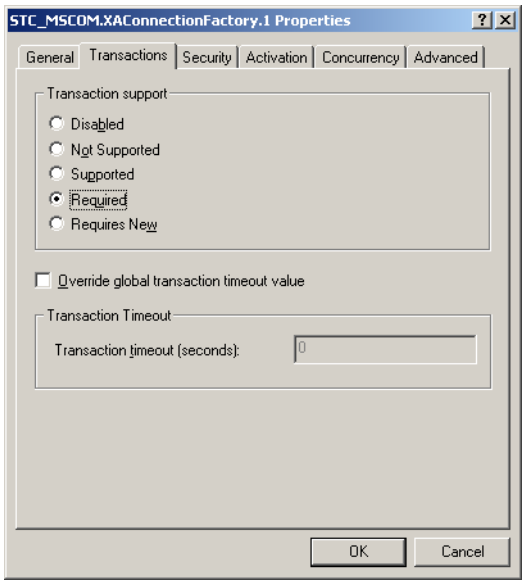
22 In the **Components** pane right-click **STC_MSCOM.XAQueueConnectionFactory** and **STC_MSCOM.XATopicConnectionFactory** (see Figure 14) and, on the shortcut menu, click **Properties**.

Figure 14 STC_MSCOM.XAConnectionFactory Properties



23 In the Transactions tab: For **Transaction support**, click **Required** (see Figure 15).

Figure 15 STC_MSCOM.XAConnectionFactory Properties:Transaction Support



24 In the Activation tab: Ensure that **Enable Just In Time Activation** is selected.

25 In the Concurrency tab: For **Synchronization support**, click **Required**, and then click **OK**.

3.1.4. Setting up the C and C++ Client

There is no specific configuration set-up that is required for the C and C++ clients. Review **Java Message Service C/C++ API Requirements** on page 12 for information regarding any specific system requirements.

When using AIX, export OBJECT_MODE=64 when using the 64 byte API Kit files or OBJECT_MODE=32 when using the 32 byte API Kit files.

For all UNIX operating systems, set the Library PATH to the **apikit\jms\c_api\lib** directory:

Table 2 Library PATHS

Operating System	Library PATH
hpux	SHLIB_PATH
sparc26	LD_LIBRARY_PATH
aix	LIBPATH
linux	LD_LIBRARY_PATH
ctru64	LD_LIBRARY_PATH

Chapter 4

Implementing JMS in Java or COM+

This chapter describes the implementation models, along with a sample implementation.

Implementing Java or COM+

- For information on what you need to use the APIs, see **Implementing JMS Models in Java or COM+** on page 28.
- For sample code, see **"Sample Code for Using JMS in Java and COM+" on page 31**.

4.1 Implementing JMS Models in Java or COM+

This section discusses how to use the Java APIs and the COM+ APIs to exchange data with an eGate system.

Considerations

To enable the client system to communicate with the eGate API Kit, you must do the following:

1. The JMS Topic/Queue names and the names of the components used must coincide.
2. Your external JMS code must know the expected data format, the name of the Topic/Queue, the name of host and port number of the JMS server.
3. The methods used must correspond to the expected data format.
4. The client code samples provided are for demonstration purposes.

4.1.1. Message Overview

The message is defined by the message structure, the header and the properties. All of the data and Events in a JMS application are expressed using messages, while the additional components exist to facilitate the transferal of messages.

Message Structure

Message Service messages are composed of the following:

- **Header** - All messages support the same set of header fields. These header fields contain values used by both clients and providers to identify and route messages.
- **Properties** - Properties provide a way to add optional header fields to messages
  - Application-specific
  - Standard properties
  - Provider-specific
- **Body** (or **Payload**) - JMS provides for supporting different types of payload. The current JMS eWay Connection supports bytes and text messaging.

Message Header Fields

When a message is received by the client, the message's header is transmitted in its entirety.

JMSDestination

The JMSDestination header field provides the destination to which the message is being sent.

JMSDeliveryMode

The JMSDeliveryMode header field provides the mode of delivery when the message was sent. The two modes of delivery are *non-persistent* and *persistent*. Non-persistent mode causes the lowest overhead, because it does not require the message to be logged to stable storage; however, non-persistent messages can be lost. Persistent mode instructs the provider to ensure that messages are not lost in transit due to provider failure.

JMSMessageID

The JMSMessageID header field contains a value intended to uniquely identify each message sent by a provider. The JMSMessageID is a String value, that should contain a unique key for identifying messages in a historical repository. The provider must provide the scope of uniqueness.

The JMSMessageID must start with the **ID:** prefix.

JMSTimestamp

The JMSTimestamp header field contains the specific time that a message is handed off to a provider to be sent. It is not the actual transmission time, because the send may occur later, due to pending transactions.

JMSExpiration

The JMSExpiration is the time that is calculated as the sum of the time-to-live value specified on the send method and the current GMT value. After the send method is returned, the message's JMSExpiration header field contains this value. If the time-to-live is specified as zero, expiration is also set to zero, and the message does not expire.

JMSRedelivered

The JMSRedelivered header filed contains the information that the message was re-delivered to the consumer. If the header is "true", the message is re-delivered, and false

if it's not. The message may be marked as re-delivered if a consumer fails to acknowledge delivery of the message, or if the JMS provider is uncertain that the consumer received the message.

```
boolean isRedelivered = message.getJMSRedelivered()
```

JMSPriority

The JMSPriority header field provides the message's priority. There is a ten-level priority value system, with 0 as the lowest priority and 9 as the highest. Priorities between 0-4 are gradations of normal priority, while 5-9 are expedited priorities.

JMSReplyTo

To enable the consumer to reply to a message associated with a specific producer, the JMSReplyTo header contains the javax.jms.Destination, indicating the address to which to reply.

```
message.setJMSReplyTo(topic);
...
Topic topic = (Topic) message.getJMSReplyTo();
```

JMSCorrelationID

The JMSCorrelationID header field provides a header field used to associate the current message with some previous message or application-specific ID. Usually the JMSCorrelationID is used to tag a message as a reply to a previous message identified by a JMSMessageID. The JMSCorrelationID can contain any value, it is not limited to JMSMessageID.

```
message.setJMSCorrelationID(identifier)
...
String correlationid = message.getJMSCorrelationID();
```

Message Properties

Properties allow a client, via message selectors, to have the JMS provider select messages based on application-specific criteria. The property values must be set prior to sending a message.

Message Body (Payload)

The full JMS specification defines six types of message body, also called *payload*. Each form is defined by a message interface. Currently, the following interfaces are supported by eGate:

- TextMessage - A message whose payload is a **java.lang.String**. It is expected that String messages will be used extensively. This type can be used to exchange both simple text messages and more complex data, such as XML documents.
- BytesMessage - A message whose payload is a stream of uninterpreted bytes. This message type is for literally encoding a body to match an existing message format. It can be used for exchanging data in an application's native format or when JMS is being used purely as a transport between two systems.

4.1.2. Creating Destinations

Destinations do not need to be created separately: they are created through the `session.createQueue()` and `session.createTopic()` functions. If these destinations do not exist, they are created automatically.

4.2 Sample Code for Using JMS in Java and COM+

Code samples are provided on the product documentation download page in the Enterprise Manager.

The external source code provided must be compiled and run, making sure that the host name and port number point to the Logical Host on which the JMS IQ Manager is running.

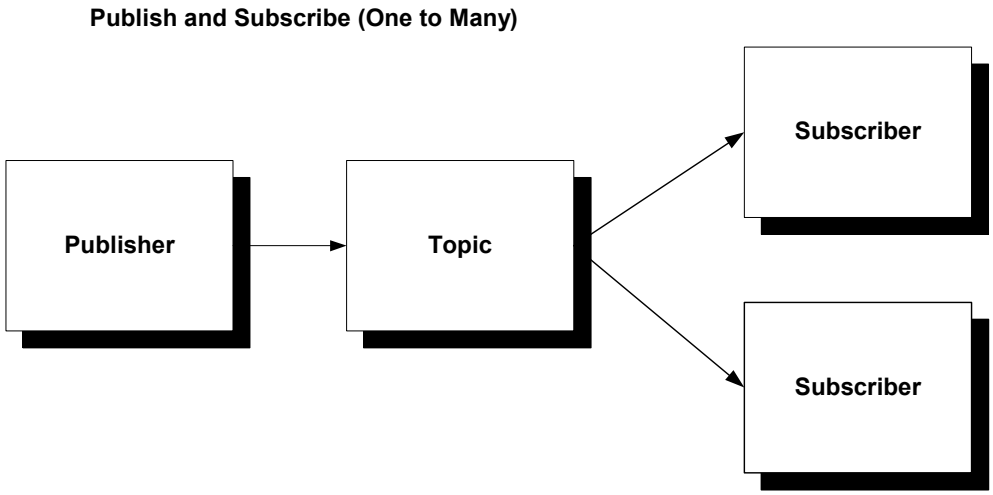
4.2.1. The Publish/Subscribe Model

The Publish/Subscribe model provides the means for a message producer or publisher, to distribute a message to one or more consumers or subscribers. There are three important points to the Publish/Subscribe model:

- Messages are delivered to consumers without having to request them. They are pushed via a channel referred to as a topic. This topic is considered a destination to which producers publish and consumers subscribe. Messages are automatically pushed to all qualified consumers.
- There is no coupling of the producers to the consumers. Both subscribers and publishers can be dynamically added at runtime, allowing the system to change as needed.
- Each client receives a copy of the messages that have been published to those topics to which it subscribes. Multiple subscribers can receive messages published by one producer.

Figure 16 below illustrates a basic Publish/Subscribe schema.

Figure 16 The Publish/Subscribe Schema



The Producer publishes a TopicA, which is stored on the Message Service. The Consumers subscribe to TopicA, which is then pushed to the consumers.

Java Publish

The code sample below illustrates the following steps:

1. Create the connection.
2. Create the session from connection (true indicates that the session is transacted).
3. Create the publisher and the bytesmessage or textmessage.
4. Send messages, varying the bytes or text if desired.
5. When all messages have been sent, close the connection.

The following code demonstrates a sample scenario using the "Publish" functionality.

```
import javax.jms.*;
import com.stc.jms.client.STCTopicConnectionFactory;

class Publisher {
    public static void main(String args[]) {
        // Read command line
        String hostName = "localhost";
        int port = 18008;
        try {
            System.out.println("-h(ost) host-name -p(ort) port-number");
            for (int i = 0; i < args.length; i++) {
                if (args[i].startsWith("-h") || args[i].startsWith("-host"))
                    hostName = args[++i];
                else if (args[i].startsWith("-p") || args[i].startsWith("-port"))
                    port = Integer.parseInt(args[++i]);
            }
        } catch (Exception e) {
            System.out.println("Error in arguments");
            System.exit(1);
        }
```

```
        // Show how to publish a message
        final String TOPICNAME = "PubSubSample";
        System.out.println("pub topic name is " + TOPICNAME);
        TopicConnection topicConnection = null;
        try {
            // Create connection, session, etc.
            TopicConnectionFactory tcf = new STCTopicConnectionFactory(hostName,
port);
            topicConnection = tcf.createTopicConnection();
            topicConnection.start();
            TopicSession topicSession = topicConnection.createTopicSession(true,
                Session.AUTO_ACKNOWLEDGE);
            Topic topic = topicSession.createTopic(TOPICNAME);
            TopicPublisher topicPublisher = topicSession.createPublisher(topic);

            // Send a msg
            TextMessage message = topicSession.createTextMessage();
            message.setText("Hello world");
            System.out.println("... Publishing message: " + message.getText());
            topicPublisher.publish(message);
            topicSession.commit();
        } catch (JMSException e) {
            System.out.println("Exception occurred: " + e.getMessage());
            e.printStackTrace();
        } finally {
            if (topicConnection != null) {
                try {
                    System.out.println("... Closing connection ...");
                    topicConnection.close();
                } catch (JMSException e) {
                    e.printStackTrace();
                }
            }
        }
    }
}
```

Java Subscribe

The code sample below illustrates the following steps:

1. Create the connection.
2. Create the session from connection (true indicates that the session is transacted).
3. Create the subscriber.
4. Register message listener (TextListener).
5. When all messages have been received, enter “Q” to quit.
6. Close the connection.

The following code sample demonstrates use of “subscribe” functionality.

```
import javax.jms.*;
import java.io.InputStreamReader;
import java.io.IOException;
import com.stc.jms.client.STCTopicConnectionFactory;

public class Subscriber {
    public static void main(String args[]) {
        // Read command line
        String hostName = "localhost";
        int port = 18008;
        try {
            System.out.println("-h(ost) host-name -p(ort) port-number");
            for (int i = 0; i < args.length; i++) {
                if (args[i].startsWith("-h") || args[i].startsWith("-host"))
```

```
                    hostName = args[++i];
                else if (args[i].startsWith("-p") || args[i].startsWith("-port"))
                    port = Integer.parseInt(args[++i]);
            }
        } catch (Exception e) {
            System.out.println("Error in arguments");
            System.exit(1);
        }


        final String TOPICNAME = "eGatePubSubSample";
        System.out.println("Topic name is " + TOPICNAME);

        TopicConnection topicConnection = null;
        try {
            // Setup connection, session, etc.
            TopicConnectionFactory qcf = new STCTopicConnectionFactory(hostName,
port);
            topicConnection = qcf.createTopicConnection();
            final TopicSession topicSession =
topicConnection.createTopicSession(true,
                Session.AUTO_ACKNOWLEDGE);
            Topic topic = topicSession.createTopic(TOPICNAME);
            TopicSubscriber topicReceiver = topicSession.createSubscriber(topic);

            // The message listener:
            topicReceiver.setMessageListener(new MessageListener() {
                public void onMessage(Message message) {
                    try {
                        if (message instanceof TextMessage) {
                            TextMessage msg = (TextMessage) message;
                            System.out.println("... Reading message: " +
                                msg.getText());
                        } else {
                            System.out.println("Message of wrong type: " +
                                message.getClass().getName());
                        }
                        topicSession.commit();
                    } catch (Exception e) {
                        System.out.println("JMSException in onMessage(): "
                            + e.toString());
                        e.printStackTrace();
                    } catch (Throwable e) {
                        System.out.println("Exception in onMessage():" +
                            e.getMessage());
                        e.printStackTrace();
                    }
                }
            });

            // Start listening
            System.out.println("Listening for messages...");
            topicConnection.start();

            // Wait until the user presses Q
            System.out.println("-----To end program, enter Q or q, then <return>");
            InputStreamReader inputStreamReader = new InputStreamReader(System.in);
            for (char answer = '\0'; Character.toUpperCase(answer) != 'Q';
                answer = (char) inputStreamReader.read()) {
            }
        } catch (IOException e) {
            System.out.println("Exception occurred: " + e.toString());
            e.printStackTrace();
        } catch (JMSException e) {
            System.out.println("Exception occurred: " + e.toString());
            e.printStackTrace();
        } finally {
            if (topicConnection != null) {
                try {
                    System.out.println("... Closing connection ...");
                    topicConnection.close();
                } catch (JMSException e) {}
```

```
            }
        }
    }
}
```

COM VB Publish/Subscribe

```
Option Explicit

Dim topicConnectionFactory As New topicConnectionFactory
Dim topicConnection As topicConnection
Dim topicSession As topicSession
Dim topic, topic2 As topic
Dim publisher As TopicPublisher
Dim subscriber As TopicSubscriber
Dim MessagePublished  As TextMessage
Dim MessageReceived As TextMessage

Private Sub Form_Load()
' You should replace the host name and port number with the actual values
    topicConnectionFactory.HostName = "localhost"
    topicConnectionFactory.Port = 24053
' Create a topic connection
    Set topicConnection = topicConnectionFactory.CreateTopicConnection()
' Create a session
    Set topicSession = topicConnection.CreateTopicSession(True,
msAutoAcknowledge)
' Start the session
    topicConnection.Start
' Create a topic
    Set topic = topicSession.CreateTopic(txtTopicName)
    Set topic2 = topicSession.CreateTopic("eGate" & txtTopicName)

' Create a publisher
    Set publisher = topicSession.CreatePublisher(topic)
' Create a subscriber
    Set subscriber = topicSession.CreateSubscriber(topic2)
End Sub

Private Sub cmdPublish_Click()
' Create a text message
    Set MessagePublished =
topicSession.CreateTextMessage(txtPublished.Text)
' Publish a message
    publisher.Publish MessagePublished
' Commit the message
    topicSession.Commit
End Sub

Private Sub cmdReceive_Click()
' Receive the message
    Set MessageReceived = subscriber.ReceiveNoWait
    If MessageReceived Is Nothing Then
        txtReceived = "No Message Received"
    Else
    ' Commit the message
        topicSession.Commit
        txtReceived = MessageReceived.Text
    End If
End Sub
```

ASP Publish

```
<%@ Language=VBScript %>


<%
          'Ensure that this page is not cached.
          Response.Expires = 0
%>

<HTML>
<HEAD>
<META NAME="GENERATOR" Content="Microsoft FrontPage 4.0">
</HEAD>
<BODY>
<%

set topicConnectionFactory =
server.CreateObject("STC_MSCOM.TopicConnectionFactory")

topicConnectionFactory.hostname = "JMS_Server_Machine"

'topicConnectionFactory.port = "JMS_Server_Port_Number"

Set topicConnection = topicConnectionFactory.CreateTopicConnection()

Set topicSession = topicConnection.CreateTopicSession(True,
AUTO_ACKNOWLEDGE)

topicConnection.Start

Set topic = topicSession.CreateTopic("test")

Set Publisher = topicSession.CreatePublisher(topic)

Set subscriber = topicSession.CreateSubscriber(topic)

Set MessagePublished = topicSession.CreateTextMessage("Hello World")

Publisher.Publish MessagePublished

topicSession.Commit

Set MessageReceived = subscriber.Receive()

topicSession.Commit

Response.write ("Answer : " & MessageReceived.Text)

%>

<P></P>

</BODY>
</HTML>
```

4.2.2. The Point-to-Point Model

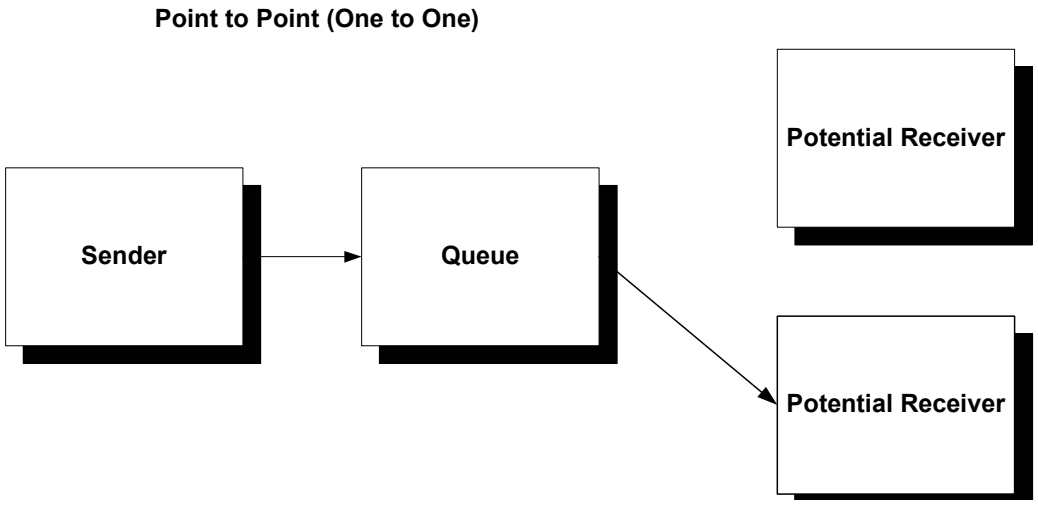
Point-to-Point messaging is based on the sending of a message to a named destination (as is the publish/subscribe model). There is no direct coupling of the producers to the consumers. One main difference between point-to-point and publish/subscribe

messaging is that in the first, messages are delivered, without consideration of the current connection status of the receiver.

In a point-to-point model, the producer is referred to as a sender, while the consumer is referred to as a receiver. The following characteristics apply:

- Message exchange takes place via a queue. The queue acts as a destination to which producers send messages, and a source from which receivers consume messages.
- Each message is delivered to only one receiver. Multiple receivers may connect to a queue, but each message in the queue may only be consumed by one of the queue's receivers.
- The queue delivers messages to consumers in the order that they were placed in the queue by the Message Service. As messages are consumed, they are removed form the "front of the line".
- Receivers and senders can be added dynamically at runtime, allowing the system to grow as needed.

Figure 17 Point to Point



Java Point-to-Point Sender

```
import javax.jms.*;
import com.stc.jms.client.STCQueueConnectionFactory;

class Sender {
    public static void main(String args[]) {
        // Read command line
        String hostName = "localhost";
        int port = 18008;
        try {
            System.out.println("-h(ost) host-name -p(ort) port-number");
            for (int i = 0; i < args.length; i++) {
               if (args[i].startsWith("-h") || args[i].startsWith("-host"))
                    hostName = args[++i];
                else if (args[i].startsWith("-p") || args[i].startsWith("-
port"))
                    port = Integer.parseInt(args[++i]);
```

```
            }
        } catch (Exception e) {
            System.out.println("Error in arguments");
            System.exit(1);
        }

        // Show how to publish a message
        final String QUEUENAME = "P2PSample";
        System.out.println("pub queue name is " + QUEUENAME);
        QueueConnection queueConnection = null;
        try {
            // Create connection, session, etc.
            QueueConnectionFactory tcf = new
STCQueueConnectionFactory(hostName, port);
            queueConnection = tcf.createQueueConnection();
            queueConnection.start();
            QueueSession queueSession =
queueConnection.createQueueSession(true,
                Session.AUTO_ACKNOWLEDGE);
            Queue queue = queueSession.createQueue(QUEUENAME);
            QueueSender producer = queueSession.createSender(queue);

            // Send a msg
            TextMessage message = queueSession.createTextMessage();
            message.setText("Hello world");
            System.out.println("... Publishing message: " +
message.getText());
            producer.send(message);
            queueSession.commit();
        } catch (JMSException e) {
            System.out.println("Exception occurred: " + e.getMessage());
            e.printStackTrace();
        } finally {
            if (queueConnection != null) {
                try {
                    System.out.println("... Closing connection ...");
                    queueConnection.close();
                } catch (JMSException e) {
                    e.printStackTrace();
                }
            }
        }
    }
}
```

Java Point-to-Point Receiver

```
import javax.jms.*;
import java.io.InputStreamReader;
import java.io.IOException;
import com.stc.jms.client.STCQueueConnectionFactory;

public class Receiver {
    public static void main(String args[]) {
        // Read command line
        String hostName = "localhost";
        int port = 18008;
        try {
            System.out.println("-h(ost) host-name -p(ort) port-number");
            for (int i = 0; i < args.length; i++) {
               if (args[i].startsWith("-h") || args[i].startsWith("-host"))
                    hostName = args[++i];
                else if (args[i].startsWith("-p") || args[i].startsWith("-
port"))
```

```
                    port = Integer.parseInt(args[++i]);
                }
            } catch (Exception e) {
                System.out.println("Error in arguments");
                System.exit(1);
            }


            final String QUEUENAME = "eGateP2PSample";
            System.out.println("Queue name is " + QUEUENAME);

            QueueConnection queueConnection = null;
            try {
                // Setup connection, session, etc.
                QueueConnectionFactory qcf = new
    STCQueueConnectionFactory(hostName, port);
                queueConnection = qcf.createQueueConnection();
                final QueueSession queueSession =
    queueConnection.createQueueSession(true,
                    Session.AUTO_ACKNOWLEDGE);
                Queue queue = queueSession.createQueue(QUEUENAME);
                QueueReceiver queueReceiver =
    queueSession.createReceiver(queue);

                // The message listener:
                queueReceiver.setMessageListener(new MessageListener() {
                    public void onMessage(Message message) {
                        try {
                            if (message instanceof TextMessage) {
                                TextMessage msg = (TextMessage) message;
                                System.out.println("... Reading message: " +
                                    msg.getText());
                            } else {
                                System.out.println("Message of wrong type: " +
                                    message.getClass().getName());
                            }
                            queueSession.commit();
                        } catch (Exception e) {
                            System.out.println("JMSException in onMessage(): "
                                + e.toString());
                            e.printStackTrace();
                        } catch (Throwable e) {
                            System.out.println("Exception in onMessage():" +
                                e.getMessage());
                            e.printStackTrace();
                        }
                    }
                });

                // Start listening
                System.out.println("Listening for messages...");
                queueConnection.start();

                // Wait until the user presses Q
                System.out.println("-----To end program, enter Q or q, then
    <return>");
                InputStreamReader inputStreamReader = new
    InputStreamReader(System.in);
                for (char answer = '\0'; Character.toUpperCase(answer) != 'Q';
                    answer = (char) inputStreamReader.read()) {
                }
            } catch (IOException e) {
                System.out.println("Exception occurred: " + e.toString());
                e.printStackTrace();
```

```
            } catch (JMSException e) {
                System.out.println("Exception occurred: " + e.toString());
                e.printStackTrace();
            } finally {
                if (queueConnection != null) {
                    try {
                        System.out.println("... Closing connection ...");
                        queueConnection.close();
                    } catch (JMSException e) {}
                }
            }
        }
    }
```

COM VB Point-to-Point

```
Option Explicit

Dim QueueConnectionFactory As New QueueConnectionFactory
Dim queueConnection As queueConnection
Dim queueSession As queueSession
Dim queue, queue2 As queue
Dim queuesender As queuesender
Dim queuereceiver, queuereceiver2 As queuereceiver
Dim MessagePublished  As BytesMessage
Dim MessageReceived As TextMessage
Dim length As Integer

Private Sub Form_Load()
' You should replace the host name and port number with the actual values
    QueueConnectionFactory.HostName = "localhost"
    QueueConnectionFactory.Port = 24053
' Create a queue Connection
    Set queueConnection = QueueConnectionFactory.CreateQueueConnection()
' Create a queue session
    Set queueSession = queueConnection.CreateQueueSession(True,
msAutoAcknowledge)
' Start the session
    queueConnection.Start
' Create a queue
    Set queue = queueSession.CreateQueue(txtQueueName)
' Create a queue sender
    Set queuesender = queueSession.CreateSender(queue)


' This is for the reply
    Set queue2 = queueSession.CreateQueue("eGate" & txtQueueName)
' Create a queue receiver
    Set queuereceiver2 = queueSession.CreateReceiver(queue2)


End Sub



Private Sub cmdSend_Click()
' Create a bytes message
    Set MessagePublished = queueSession.CreateBytesMessage
    MessagePublished.ClearBody
    length = Len(txtPublished.Text)
    MessagePublished.WriteBytes txtPublished.Text
' Send this message
    queuesender.Send MessagePublished
' Commit this message
```

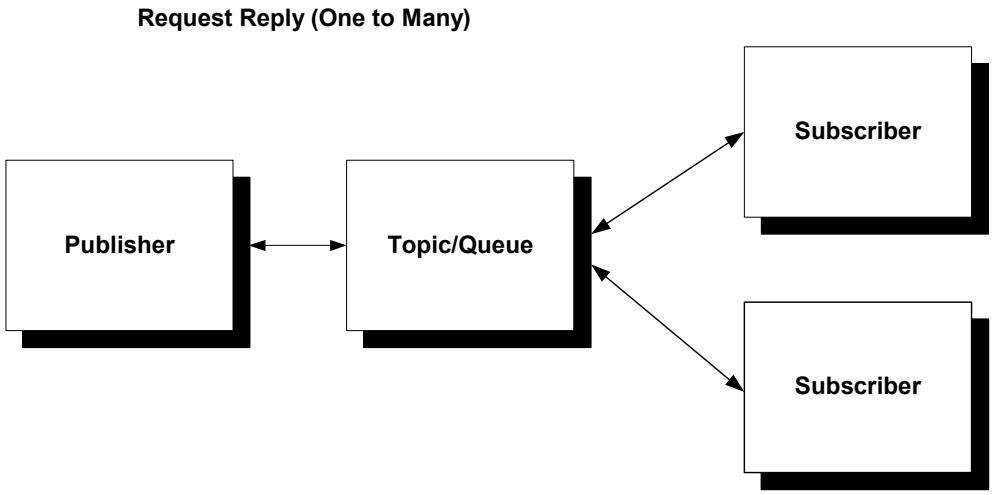
```
    queueSession.Commit
End Sub

Private Sub cmdReceive_Click()
' Receive the message
    Set MessageReceived = queuereceiver2.ReceiveNoWait
    If MessageReceived Is Nothing Then
        txtReceived = "No Message Received"
    Else
    ' Commit this message
        queueSession.Commit
        Dim data As Variant
        txtReceived.Text = MessageReceived.Text
    End If
End Sub
```

4.2.3. The Request-Reply Model

JMS provides the JMSReplyTo message header field for specifying the destination to which the reply to a message is to be sent. The JMSCorrelationID header field of the reply can be used to reference the original request. Temporary queues and topics can be used as unique destinations for replies. It can be implemented so that one message yields one reply, or one message yields many replies.

Figure 16 below illustrates a basic Request-Reply schema.

Figure 18 The Request-Reply Schema



1. A request is received by the **JMS Connection,** which is controlled by the JMS IQ Manager, the JMSReplyTo property is read into the internally directed by the Collaboration.
2. eGate reads in the request from **SampleTopicRequestor**, and appends a message to the end of the message for verification's sake.
3. The **SeeBeyond JMS IQ Manager** sends the message to a Temporary Topic via the JMS Connection.

4 The reply subscriber receives the message.

5 When the **Message Service** users disconnect, the temporary topic is destroyed.

The scenario discussed above need not be configured exactly in this manner. This is just an example that demonstrates a possible scenario.

Java Request-Reply

The code sample below illustrates the following steps:

1 Create the connection.

2 Create the session from connection (true indicates that the session is transacted).

3 Create the topic/queue and byte or text message.

4 Send messages, varying the bytes or text if desired.

5 When all messages have been sent, close the connection.

Java TopicRequestor

```
import javax.jms.*;
import com.stc.jms.client.STCTopicConnectionFactory;
import java.io.*;

class SampleTopicRequestor {
    public static void main(String args[]) {
        String usage = "Usage: java SampleTopicRequestor [-f/-m file/message] " +
            "[-topic topic] [-host host] [-port port]" +
            "[-help] ";
        String help = usage + "\n\n" +
            "-f <file name>\n" +
            "-m <message: default SampleMessage>\n" +
            "-topic <topic name: default TopicRequestor>\n" +
            "-host <host name where ms server is running: default localhost>\n" +
            "-port <port where ms server is running: default 24053>\n" +
            "-help <this screen>\n";

        TopicConnection requestConnection = null;

        try {
            // Read command line
            String topicName = "TopicRequestorSample";
            String messageToSend = "SampleMessage";
            String host = "localhost";
            int port = 18008;

            for (int i = 0; i < args.length; i++) {
                if (args[i].equals("-f")) {
                    File thisFile = new File(args[++i]);
                    BufferedReader stream = null;
                    try {
                        int fileLength = (int) thisFile.length();
                        char[] myCharMessage = new char[fileLength + 1];
                        stream = new BufferedReader(new InputStreamReader(new
                            FileInputStream(thisFile)));
                        stream.read(myCharMessage, 0, fileLength);
                        messageToSend = new String(myCharMessage);
                    } finally {
                        stream.close();
                    }
                } else if (args[i].equals("-m"))
                    messageToSend = args[++i];
                else if (args[i].equals("-topic"))
                    topicName = args[++i];
```

```
                else if (args[i].equals("-host"))
                    host = args[++i];
                else if (args[i].equals("-port"))
                    port = Integer.parseInt(args[++i]);
                else if (args[i].equals("-help")) {
                    System.out.println(help);
                    System.exit(0);
                } else {
                    System.out.println(usage);
                    System.exit(1);
                }
            }

            // Create TopicConnection, session, etc
            TopicConnectionFactory factory = new STCTopicConnectionFactory(host,
port);
            requestConnection = factory.createTopicConnection();
            requestConnection.setExceptionListener(new ExceptionListener() {
                public void onException(JMSException e) {
                    e.printStackTrace();
                }
            });
            TopicSession topicSession = requestConnection.createTopicSession(false,
                Session.AUTO_ACKNOWLEDGE);
            Topic topic = topicSession.createTopic(topicName);
            TopicRequestor requestor = new TopicRequestor(topicSession, topic);
            requestConnection.start();

            // Create request and wait for reply
            System.out.println("Sending request...");
            TextMessage request = topicSession.createTextMessage(messageToSend);
            TextMessage reply = (TextMessage) requestor.request(request);
            System.out.println("... Got reply: " + reply.getText());
        } catch (Throwable e) {
            e.printStackTrace();
        } finally {
            if (requestConnection != null) {
                try {
                    requestConnection.close();
                } catch (Exception ignore) {
                }
            }
        }
    }
}
```

Java QueueRequestor

```
import javax.jms.*;
import com.stc.jms.client.STCQueueConnectionFactory;
import java.io.*;

class SampleQueueRequestor {
    public static void main(String args[]) {
        String usage = "Usage: java SampleQueueRequestor [-f/-m
file/message] " +
            "[-queue queue] [-host host] [-port port]" +
            "[-help] ";
        String help = usage + "\n\n" +
            "-f <file name>\n" +
            "-m <message: default SampleMessage>\n" +
            "-queue <queue name: default QueueRequestor>\n" +
            "-host <host name where ms server is running: default
localhost>\n" +
            "-port <port where ms server is running: default 24053>\n" +
            "-help <this screen>\n";

        QueueConnection requestConnection = null;
```

```
        try {
            // Read command line
            String queueName = "QueueRequestorSample";
            String messageToSend = "SampleMessage";
            String host = "localhost";
            int port = 18008;

            for (int i = 0; i < args.length; i++) {
                if (args[i].equals("-f")) {
                    File thisFile = new File(args[++i]);
                    BufferedReader stream = null;
                    try {
                        int fileLength = (int) thisFile.length();
                        char[] myCharMessage = new char[fileLength + 1];
                      stream = new BufferedReader(new InputStreamReader(new
                            FileInputStream(thisFile)));
                        stream.read(myCharMessage, 0, fileLength);
                        messageToSend = new String(myCharMessage);
                    } finally {
                        stream.close();
                    }
                } else if (args[i].equals("-m"))
                    messageToSend = args[++i];
                else if (args[i].equals("-queue"))
                    queueName = args[++i];
                else if (args[i].equals("-host"))
                    host = args[++i];
                else if (args[i].equals("-port"))
                    port = Integer.parseInt(args[++i]);
                else if (args[i].equals("-help")) {
                    System.out.println(help);
                    System.exit(0);
                } else {
                    System.out.println(args[i]);
                     System.out.println(usage);
                    System.exit(1);
                }
            }

            // Create QueueConnection, session, etc
            QueueConnectionFactory factory = new
STCQueueConnectionFactory(host, port);
            requestConnection = factory.createQueueConnection();
           requestConnection.setExceptionListener(new ExceptionListener() {
                public void onException(JMSException e) {
                    e.printStackTrace();
                }
            });
            QueueSession queueSession =
requestConnection.createQueueSession(false,
                    Session.AUTO_ACKNOWLEDGE);
            Queue queue = queueSession.createQueue(queueName);
            QueueRequestor requestor = new QueueRequestor(queueSession,
queue);
            requestConnection.start();

            // Create request and wait for reply
            System.out.println("Sending request...");
            TextMessage request =
queueSession.createTextMessage(messageToSend);
            TextMessage reply = (TextMessage) requestor.request(request);
            System.out.println("... Got reply: " + reply.getText());
        } catch (Throwable e) {
```

```
            e.printStackTrace();
        } finally {
            if (requestConnection != null) {
                try {
                    requestConnection.close();
                } catch (Exception ignore) {
                }
            }
        }
    }
}
```

COM VB TopicRequestor

```
Option Explicit

Dim topicConnectionFactory As New topicConnectionFactory
Dim topicConnection As topicConnection
Dim topicSession As topicSession
Dim topic As topic
Dim topicRequestor As New topicRequestor
Dim MessagePublished  As TextMessage
Dim MessageReceived  As TextMessage

Private Sub Form_Load()
' You should replace the host name and port number with the actual values
    topicConnectionFactory.HostName = "localhost"
    topicConnectionFactory.Port = 24053
' Create a topic connection
    Set topicConnection = topicConnectionFactory.CreateTopicConnection()
' Create a topic session
    Set topicSession = topicConnection.CreateTopicSession(False,
msAutoAcknowledge)
' Start the session
    topicConnection.Start
End Sub

Private Sub cmdStart_Click()
    cmdStart.Enabled = False
' Create a topic
    Set topic = topicSession.CreateTopic(txtTopicName)
' Create a topic requestor
    topicRequestor.Create topicSession, topic
' Create a text message
    Set MessagePublished =
topicSession.CreateTextMessage(txtPublished.Text)
' Request a message
    Set MessageReceived = topicRequestor.Request(MessagePublished)
    txtReceived = MessageReceived.Text
    cmdStart.Enabled = True
End Sub
```

COM VB QueueRequestor

```
Option Explicit

Dim queueConnectionFactory As New queueConnectionFactory
Dim queueConnection As queueConnection
Dim queueSession As queueSession
Dim queue As queue
Dim queueRequestor As New queueRequestor
Dim MessagePublished  As TextMessage
Dim MessageReceived  As TextMessage
```

```
Private Sub Form_Load()
' You should replace the host name and port number with the actual
' values
    queueConnectionFactory.HostName = "localhost"
    queueConnectionFactory.Port = 24053
' Create a queue connection
    Set queueConnection =
queueConnectionFactory.CreateQueueConnection()
' Create a queue session
    Set queueSession = queueConnection.CreateQueueSession(False,
msAutoAcknowledge)
' Start the session
    queueConnection.Start
End Sub

Private Sub cmdStart_Click()
' Create a queue
    cmdStart.Enabled = False
    Set queue = queueSession.CreateQueue(txtQueueName)
' Create a text message
    Set MessagePublished =
queueSession.CreateTextMessage(txtPublished.Text)
' Create a queue requestor
    queueRequestor.Create queueSession, queue
' Request a message
    Set MessageReceived = queueRequestor.Request(MessagePublished)
    txtReceived = MessageReceived.Text
    cmdStart.Enabled = True
End Sub
```

4.2.4. JNDI

Naming Operations

JNDI can be used to perform various naming operations. The most common operations are:

- Looking up an object
- Listing the contents of a context
- Adding, overwriting, and removing a binding
- Renaming an object
- Creating and destroying sub-context

To use JNDI in your programs, you need to set up its compilation and execution environments.

Applications that need to connect to the JNDI service of the Integration Server should specify the following for the initial context Factory:

```
com.stc.is.naming.NamingContextFactory.
```

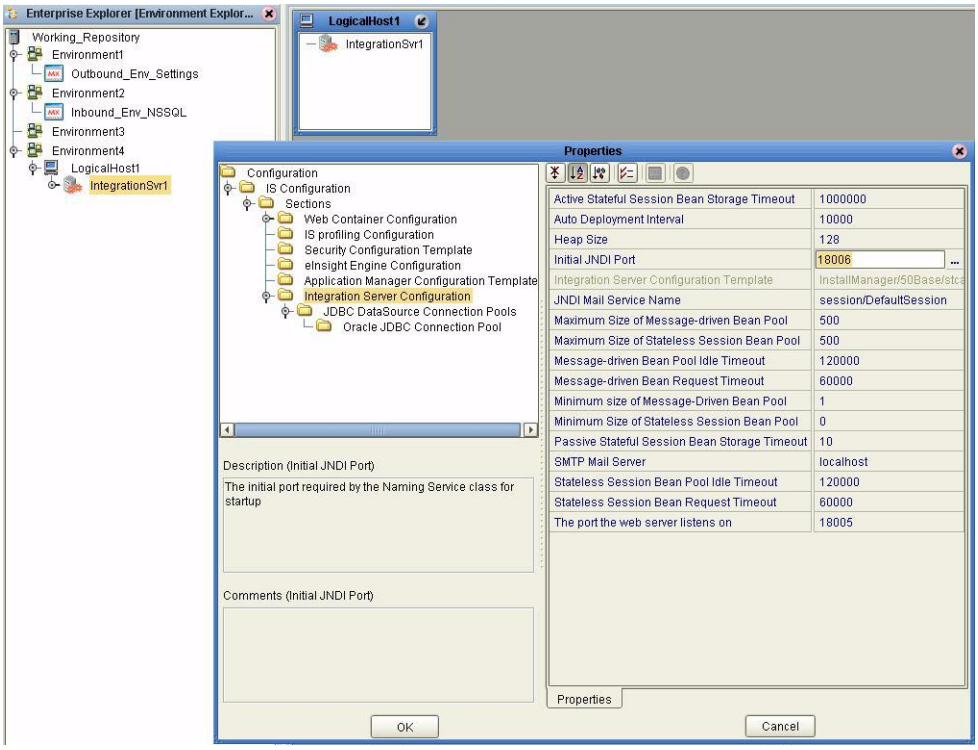
The URL appears as follows:

```
java://<server name>:<port>
```

The server name is the name of the machine the JNDI service (the Integration Server) is running on. The port number can be found in the properties screen of the Integration Server in Enterprise Designer's Environment tab:

1. Click the Environment Explorer tab. On the shortcut menu, click New Logical Host.
2. On the Environment Explorer, select the LogicalHost node and right click. On the shortcut menu click New SeeBeyond Integration Server.
3. On the Environment Explorer, select the IntegrationSrv node and right click. On the shortcut menu click Properties.
4. On the Properties window, expand the IS Configuration node selecting Integration Server Configuration.

Figure 19 Integration Server Properties



The following code fragment illustrates how to create an initial context:

```
// Connect JDNI
Properties p = new Properties();
p.put(Context.INITIAL_CONTEXT_FACTORY,
    "com.stc.is.naming.NamingContextFactory");
p.put(Context.PROVIDER_URL, "java://blue:18006");
InitialContext ctx = new InitialContext(p);
```

In this example, "blue" is the name of the machine that the JNDI Service (the Integration Server) is running on.

4.2.5. JNDI names

The naming convention within the ICAN suite for connection factories is as follows:

```
jms/connectionfactory/<domain>/<logicalhostname>_<message server
name>
```

Here <domain> is one of the four following literals:

queue

topic

xa_queue

xa_topic

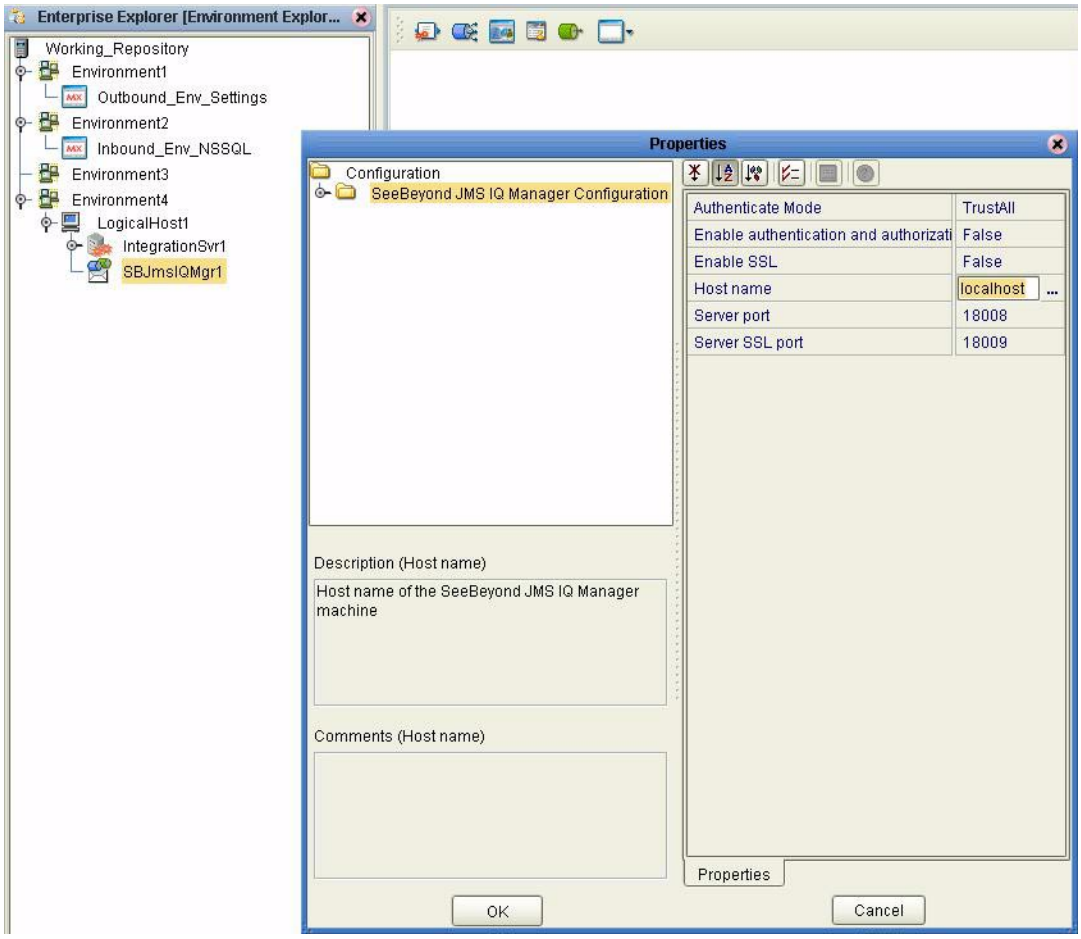
Example: this is the name used to look up a topic connection factory for the JMS IQ Manager named "MessageSvr1" that exists within the logical host named "LogicalHost1".

```
jms/connectionfactory/topic/LogicalHost1_MessageSvr1
```

Running the application and JMS IQ Manager on different machines

Note that by default the server name of JMS IQ Manager is "LogicalHost". This works if the Integration Server and the JMS IQ Manager are on the same machine. It is this server name that is used in the connection factories that are bound into JNDI. If an application that wants to use these connections factories runs on a different machine than the machine the JMS IQ Manager is running on, the LogicalHost of course can not be used to connect to JMS IQ Manager. In those cases it is important to specify the server name in the properties screen of the JMS IQ Manager in Enterprise Designer.

Figure 20 SeeBeyond JMS IQ Manager Properties



The following **.jar** files are required to run the sample:

Table 3 API Kit Required **.jar** Files

Name of Jar File	Used During Compile TIme	Used During Run Time
jms.jar	Yes	Yes
jta.jar	No	No
log4j.jar (required for Integration Server only)	No	Yes
com.stc.jmsis.jar	No	Yes

Make and Create a Connection

The following code fragment illustrates how to look up a connection factory, make a connection and create a connection.

```
import java.util.Properties;
import javax.jms.*;
import javax.naming.Context;
import javax.naming.InitialContext;
```

```
public class Sample {
  public static void main(String[] args) {
    try {
      // Connect JDNI
      Properties p = new Properties();
      p.put(Context.INITIAL_CONTEXT_FACTORY,
        "com.stc.is.naming.NamingContextFactory");
      p.put(Context.PROVIDER_URL, "java://blue:18006");
      InitialContext ctx = new InitialContext(p);
// Find connection factory
      TopicConnectionFactory fact = (TopicConnectionFactory)
ctx.lookup(
        "jms/connectionfactory/topic/LogicalHost1_SBJmsIQMgr1");
// Create connection, session
      TopicConnection conn = fact.createTopicConnection();
      TopicSession session = conn.createTopicSession(false,
        Session.AUTO_ACKNOWLEDGE);
// Destination (can also be done through JNDI but requires external
binding)
      // Queue dest = (Queue) ctx.lookup("java:/jms/...");
      Topic dest = session.createTopic("Topic1");
// Send msg
      TopicPublisher producer = session.createPublisher(dest);
      TextMessage msg1 = session.createTextMessage("Hello world!");
      producer.publish(msg1, DeliveryMode.NON_PERSISTENT,
        Message.DEFAULT_DELIVERY_MODE, 0);
      conn.close();
      System.out.println("Message published");
    } catch (Exception ex) {
      ex.printStackTrace();
    }
  }
}
```

To compile:

```
javac -classpath jms.jar Sample.java
```

To run:

```
java -cp com.stc.jmsis.jar;.;jms.jar;log4j.jar Sample
```

4.2.6. Registering Connection Factories in the JNDI File Provider

To register the JNDI using the *File Provider* available from Sun:

```
// Setup JNDI
Properties props = new Properties();
props.put(Context.INITIAL_CONTEXT_FACTORY,
"com.sun.jndi.fscontext.RefFSContextFactory");
props.put(Context.PROVIDER_URL, "file:///stcms");
InitialContext ctx = new InitialContext(props);
// Bind factories
ctx.rebind("queue", new STCQueueConnectionFactory("localhost",
18006));
ctx.rebind("topic", new STCTopicConnectionFactory("localhost",
18006));
ctx.rebind("xaqueue", new STCXAQueueConnectionFactory("localhost",
18006));
ctx.rebind("xatopic", new STCXATopicConnectionFactory("localhost",
18006));
ctx.close();
```

To compile the code:

```
javac -classpath com.stc.jmsis.jar;jms.jar Sample.java
```

To run:

```
java -cp com.stc.jmsis.jar;.;jms.jar;fscontext.jar;providerutil.jar
Sample
```

Table 4 Registering JNDI

Jar Files	Required at Compile Time	Required at Run Time
jms.jar	Yes	Yes
jta.jar	No	No
com.stc.jmsis.jar	Yes	Yes
fscontext.jar	No	Yes
providerutil.jar	No	Yes

The following sample illustrates how to use the bindings file:

```
import java.util.Properties;
import javax.jms.*;
import javax.naming.Context;
import javax.naming.InitialContext;
public class Sample {
  public static void main(String[] args) {
    try {
      // Setup JNDI
      Properties props = new Properties();
      props.put(Context.INITIAL_CONTEXT_FACTORY,
            "com.sun.jndi.fscontext.RefFSContextFactory");
      props.put(Context.PROVIDER_URL, "file:.");
      InitialContext ctx = new InitialContext(props);
// Find connection factory
      TopicConnectionFactory fact = (TopicConnectionFactory)
ctx.lookup(
        "topic");
// Create connection, session
      TopicConnection conn = fact.createTopicConnection();
      TopicSession session = conn.createTopicSession(false,
        Session.AUTO_ACKNOWLEDGE);
// Destination (can also be done through JNDI but requires external
binding)
      // Queue dest = (Queue) ctx.lookup("java:/jms/...");
      Topic dest = session.createTopic("Topic1");
// Send msg
      TopicPublisher producer = session.createPublisher(dest);
      TextMessage msg1 = session.createTextMessage("Hello world!");
      producer.publish(msg1, DeliveryMode.NON_PERSISTENT,
        Message.DEFAULT_DELIVERY_MODE, 0);
      conn.close();
      System.out.println("Message published");
    } catch (Exception ex) {
      ex.printStackTrace();
    }
  }
}
```

To compile:

```
javac -classpath jms.jar Sample.java
```

To run:

```
java -cp com.stc.jmsis.jar;.;jms.jar;fscontext.jar;providerutil.jar
Sample
```

Table 5 Jar File for Binding

Jar Files	Required at Compile Time	Required at Run Time
jms.jar	Yes	Yes
jta.jar	No	No
com.stc.jmsis.jar	No	Yes
fscontext.jar	No	Yes
provideruti.jar	No	Yes

4.2.7. Registering JNDI Using the JNDIRegister Tool

To bind the ConnectionFactories from the command line:

The JMS IQ Manager also has functionality built in to create ConnectionFactories and bind them into any JDNI provider. This functionality can be invoked from the command line. The class com.stc.jms.client.JNDIRegister utility class is located in the com.stc.jmsis.jar. To use this utility do the following:

```
Usage: JNDIRegister -fact InitialCtxFactory -url ProviderUrl -type
Object type
      -jndiname JNDI name to be used to register the object
      [-dest Destination name] [-host host-name] [-port port]


Options/Arguments:
       InitialCtxFactory       *Initial context factory class, e.g.
com.sun
.jndi.fscontext.RefFSContextFactory
       ProviderUrl             *Url to access the JNDI provider URL,
e.g. file://stcms.jndi
       Object type             *Type: Q=queue connection factory;
q=queue;
                               T=topic connection factory; t=topic
       JNDI name to be used to register the object
                              *Name that the object will be registered
under
       -dest Destination name  Name of queue or topic
       -host host-name         specifies host name
       -port port              specifies server port number

* Required
```

To register a queue

```
java -cp com.stc.jmsis.jar;jms.jar;fscontext.jar;providerutil.jar
-fact com.sun.jndi.fscontext.RefFSContextFactory -url file://c:\temp
-type q -jndiname jndi-myqueue -dest myqueue
```

To register a topic

```
java -cp com.stc.jmsis.jar;jms.jar;fscontext.jar;providerutil.jar
-fact com.sun.jndi.fscontext.RefFSContextFactory -url file://c:\temp
-type t -jndiname jndi-mytopic -dest mytopic
```

To register a connection factory

```
java -cp com.stc.jmsis.jar;jms.jar;fscontext.jar;providerutil.jar
-fact com.sun.jndi.fscontext.RefFSContextFactory -url file://c:\temp
-type Q -jndiname jndi-myqfact -h 127.0.0.1 -p 7125
```

4.2.8. Creating ConnectionFactories

To create connection factories for JMS IQ Manager directly, create instances of one of the following classes:

- com.stc.jms.client.STCQueueConnectionFactory
- com.stc.jms.client.STCTopicConnectionFactory
- com.stc.jms.client.STCXAQueueConnectionFactory
- com.stc.jms.client.STCXATopicConnectionFactory

These classes come with several different constructors. The two most useful constructors are:

```
public STCConnectionFactory(String host, int port)
```

and

```
public STCConnectionFactory(Properties p)
```

The latter constructor allows a number of advanced properties to be set. These properties are as follows:

`Property name: com.stc.jms.sockets.ServerHost`

Specifies the host name of the machine the message server is running on.

`Property name: com.stc.jms.sockets.ServerPort`

Specifies the port number that the message server is listening on.

`Property name: com.stc.jms.sockets.ConnectionManager.class`

Specifies which threading model will be used for asynchronous communication. Valid values are "com.stc.jms.sockets.MuxConnectionMgr" and "com.stc.jms.sockets.ThreadPerConnectionMgr" (default). The former will use one thread for all socket connections; the second will use one thread for each socket. Note that there will be one socket in use for each consumer; when consumers are used with a message listener, or are used with a receive(timeout), asynchronous communication is used. Using the "MUX"-model reduces the number of threads, but is more CPU intensive because there will be one thread polling sockets for incoming data.

`Property name: com.stc.jms.sessionpooling`

Specifies whether session pooling is used. Valid values are true and false (default). When session pooling is used, sessions are not physically closed when the close() method on a session is called. Instead they are returned to a pool; when a new session needs to be created, an attempt is made to reuse a session from the pool. Because a separate socket connection is used for each session, session pooling will speed up performance in situations where the application is creating and closing sessions rapidly. Note that sessions are pooled per connection, i.e. the pool of sessions is not global but each connection has its own session pool. When sessions

are pooled, producers are also pooled automatically, i.e. when the close() method on a producer is called, the producer is not closed but is returned to the session's producer pool. When a new producer is created, an attempt is made to satisfy this request using the producer pool. Because each producer uses a socket connection, producer pooling improves performance in applications that rapidly create and close producers. Sessions and producers are closed physically, when the JMS connection that created them is closed physically.

`Property name: com.stc.jms.connectionpooling`

Specifies whether connection pooling is used. Valid values are true and false (default). When connection pooling is used, a connection is not physically closed when the close() method on a connection is called. Instead, the connection is returned to a global pool. Connection pooling is useful only in combination with session pooling.

`Property name: com.stc.jms.connectionpooling.maxidlesec`

Specifies the number of seconds that idle JMS connections will reside in the global connection pool before they are closed by a subsequent connection request. The default is 30 seconds.

`Property name: com.stc.jms.ssl.authenticationmode`

Determines if TCP/IP connections made by the client to the server will use SSL, and if so, how the server will be authenticated. Valid values are Authenticate, TrustAll and False (default). If no value is specified, or if False is specified, connections will not use SSL. A value of Authenticate will cause the client to assure that the certificate presented by the JMS server is trusted, i.e. is signed by an authority present in the truststore. See below on how to specify the truststore. A value of TrustAll will cause the client to accept any certificate presented by the JMS server. Note: if there is no value specified for this property, it will be looked up in the System properties.

`Property name: javax.net.ssl.trustStore`

Specifies the path to the store that contains trusted certificate authorities. This store is used to authenticate the JMS server when using SSL and the Authenticate authentication mode. Note: if there is no value specified for this property, it will be looked up in the System properties.

`Property name: javax.net.ssl.trustStorePassword`

Specifies the password used to protect the certificate store that contains trusted certificate authorities. This store is used to authenticate the JMS server when using SSL and the Authenticate authentication mode. Note: if there is no value specified for this property, it will be looked up in the System properties.

`Property name: security.provider.1`

Specifies the security provider used for SSL (default: com.sun.net.ssl.internal.ssl.Provider). Note that if the provider is set to com.ibm.jsse.IBMJSSEProvider, the IbmX509 TrustManagerFactory will be used; in other cases, the SunX509 TrustManagerFactory will be used. Note: if there is no value specified for this property, it will be looked up in the System properties.

JNDI Samples

This sample requires edits be made to generic information. See the code samples below in **bold** typeface.

To run, you need to set up the classpath to include the file system service provider classes (**fscontext.jar** and **providerutil.jar**). (Samples can be downloaded from **http://java.sun.com/products/jndi/tutorial/getStarted/examples/naming.html**). For this example the following was used, and should be modified to suit your needs:

```
set CLASSPATH=%CLASSPATH%;fscontext.jar;providerutil.jar
```

Each of the following samples requires the following imports:

```
import javax.jms.*;
import javax.naming.*;
import java.util.Properties;
```

A sample class is declared:

```
public class queuereply
{
    public static void main( String[] args )
    {
    try {
```

The definition of JNDI properties is made here, but could also be made using `jndi.properties` file:

```
// JNDI parameters - you will probably use jndi.properties with values
//specific to your own JNDI provider.
//
        Properties env = new Properties();
        env.put(Context.INITIAL_CONTEXT_FACTORY,
            "com.sun.jndi.fscontext.RefFSContextFactory");
        env.put(Context.PROVIDER_URL, "file:/tmp/tutorial");
        Context jndi = new InitialContext(env);
```

QueueConnectionFactory Sample

In addition to the above, the following is included in the QueueConnectionFactory sample:

```
// Instantiate a SeeBeyond QueueConnectionFactory object and bind to
// JNDI
        STCQueueConnectionFactory qcf = new
STCQueueConnectionFactory("myhostname", 24056);
        try {
        jndi.bind("QueueConnectionFactory", (QueueConnectionFactory)
qcf);
        }
        catch (javax.naming.NameAlreadyBoundException e) {
        }
```

Once the `bind` has been established, a `NameAlreadyBoundException` will be returned if bind is called again. `rebind` is used to overwrite the binding.

```
// Lookup the object in JNDI, and print some info for verification
        Object obj = jndi.lookup("QueueConnectionFactory");
        qcf = (STCQueueConnectionFactory) obj;
```

```
        System.out.println ("Looked up QueueConnectionFactory host:"
+ qcf.getHost());
        System.out.println ("Looked up QueueConnectionFactory port:"
+ qcf.getPort());
```

TopicConnectionFactory Sample

In addition to the above, the following is included in the TopicConnectionFactory sample:

```
// Instantiate a SeeBeyond TopicConnectionFactory object and bind to
// JNDI
        STCTopicConnectionFactory tcf = new
STCTopicConnectionFactory("anotherhost", 24053);
        try {
        jndi.rebind("TopicConnectionFactory",
(TopicConnectionFactory) tcf);
        }
        catch (javax.naming.NameAlreadyBoundException e) {
        }

// Lookup the object in JNDI, and print some info for verification
        obj = jndi.lookup("TopicConnectionFactory");
        tcf = (STCTopicConnectionFactory) obj;
        System.out.println ("Looked up TopicConnectionFactory host:"
+ tcf.getHost());
        System.out.println ("Looked up TopicConnectionFactory port:"
+ tcf.getPort());
```

Topic Sample

In addition to the above, the following is included in the Topic sample:

```
// Instantiate a SeeBeyond Topic object and bind to JNDI
        STCTopic top = new STCTopic("AccountsPayableTopic");
        try {
        jndi.bind("APTopic", (Topic) top);
        }
        catch (javax.naming.NameAlreadyBoundException e) {
        }

// Lookup the object in JNDI, and print some info for verification
        obj = jndi.lookup("APTopic");
        s = new String(obj.getClass().getName());
        System.out.println ("APTopic class:"+s);

    } catch( Exception e ) {
        e.printStackTrace();
    }
    }
}
```

At this point the retrieved objects can now communicate with the SeeBeyond Message Service and eGate.

4.2.9. The Message Selector

A message selector allows a client to specify, via the message header, those messages in which the client is interested. Only messages for which the headers and properties match the selector are delivered. The semantics of not delivered differ depending on the

MessageConsumer implemented. Message selectors cannot reference message body values.

The message selector matches a message, provided the selector evaluates to "true", when the message's header field and the property values are substituted for the corresponding identifiers within the selector.

For more information about Message Selection, see chapter *3.8, Message Selection*, of the *Java Message Service Version 1.0.2.b* available at:

http://java.sun.com/products/jms/docs.html

Message Selector Syntax

A message selector is defined as a String, wherein the syntax is composed, according to a subset of the SQL92* conditional expression syntax. If the value of a message selector is provided as an empty string, the value is treated as "null" and indicates that there is no message selector for the message consumer.

The order of evaluation for message selectors is from left to right within precedence level. Parentheses can be used to change this order. Predefined selector literals and operator names are written here in upper case; however, they are case-insensitive.

Identifiers

An identifier is that part of the expression that provides the information by which to make the comparison. For example, the identifiers in the following expression are Age, Weight, and LName:

```
Age < 35 AND Weight >= 110.00 and LName = `Jones'
```

Identifiers can be any application-defined, JMS-defined, or provider-specific property, or one of several JMS headers. Identifiers must match the property or JMS header name exactly (they are also case sensitive).

The following JMS headers can be used as identifiers:

- JMSDeliveryMode
- JMSPriority
- JMSMessageID
- JMSTimestamp
- JMSCorrelationID
- JMSType

The following JMS headers cannot be used as identifiers because their corresponding values are Destination objects, whose underlying value is proprietary, and therefore undefined:

- JMSDestination
- JMSReplyTo

The JMSRedelivered value may be changed during delivery. For example, if a consumer uses a message selector where "JMSRedelivered = FALSE", and there was a failure delivering a message, the JMSRedelivered flag might be set to TRUE. JMSExpiration is

not supported as an identifier, because JMS providers may choose to implement this value in different manners. (Some may store it with the message, while others calculate it as needed.)

Literals

Expression values that are hard-coded into the message selector are referred to as literals. In the message selector shown here, 35, 110.00, and 'Jones' are all literals:

```
Age < 35 AND Weight >= 110.00 AND LName = 'Jones'
```

String literals are enclosed in single quotes. An apostrophe or single quote can be included in a String literal by using two single quotes. For example: O''Leary''s

Numeric literals are expressed using exact numerical (+35, 30, -450), approximate numerical with decimal (-22.33, 110.00, +8.0), or scientific (-7E4) notation.

Declaring a Message Selector

When a consumer is created to implement a message selector, the JMS provider must validate that the selector statement is syntactically correct. If the selector is not correct, the javax.jms.InvalidSelectorException is thrown.

```
protected void writeMessage(String text) throws JMSException{
    TextMessage message = session.createTextMessage();
    message.setText(text);
    message.setStringProperty("username",username);
    publisher.publish(message);
}
```

JMS clients would then use that property to filter messages. Message selectors are declared when the message consumer is created:

```
TopicSubscriber subscriber =
session.createSubscriber(xTopic, "username <> 'John' ", false);
```

The message selector ("username <> 'John' ") tells the Message Service to deliver to the consumer only those messages that do NOT have the username property equal to 'John'.

Java Message Selector Publisher

```
import javax.jms.*;
import com.stc.jms.client.STCTopicConnectionFactory;

public class SelectorPublisher {
    private static final String HOST = "localhost";
    private static final int PORT = 18008;
    private static final String PROP_NAME = "property";
    private static final String TOPIC_NAME = "Selector";

    public static void main(String[] args) {
        // Read command line
        String hostName = HOST;
        int port = PORT;
        try {
            System.out.println("-h(ost) host-name -p(ort) port-number");
            for (int i = 0; i < args.length; i++) {
               if (args[i].startsWith("-h") || args[i].startsWith("-host"))
                    hostName = args[++i];
                else if (args[i].startsWith("-p") || args[i].startsWith("-
port"))
                    port = Integer.parseInt(args[++i]);
```

```
            }
        } catch (Exception e) {
            System.out.println("Error in arguments");
            System.exit(1);
        }

        TopicConnection conn = null;
        try {
            // Create connection, session, etc.
            TopicConnectionFactory factory = new STCTopicConnectionFactory(
                hostName, port);
            conn = factory.createTopicConnection();
            TopicSession topicSession = conn.createTopicSession(true,
                Session.AUTO_ACKNOWLEDGE);
            Topic topic = topicSession.createTopic(TOPIC_NAME);
            TopicPublisher publisher = topicSession.createPublisher(topic);
            publisher.setDeliveryMode(DeliveryMode.PERSISTENT);

            // Publish 10 messages
            for (int i = 0; i < 10; i++) {
                TextMessage msg = topicSession.createTextMessage();
                int index = i % 10;
                msg.setStringProperty(PROP_NAME, "" + index);
                msg.setText("This is the message body.");
                publisher.publish(msg);
                System.out.println("... Published 1 message with " +
PROP_NAME
                    + " = " + i);
            }
            topicSession.commit();
        } catch (Exception ex) {
            ex.printStackTrace();
        } finally {
            if (conn != null) {
                try {
                    conn.close();
                } catch (Exception ignore) {
                }
            }
        }
    }
}
```

Java Message Selector Subscriber

```
import javax.jms.*;
import com.stc.jms.client.STCTopicConnectionFactory;

public class SelectorSubscriber {
    private static final String HOST = "localhost";
    private static final int PORT = 18008;
    private static final String PROP_NAME = "property";
    private static final String TOPIC_NAME = "Selector";

    public static void main(String[] args) {
        // Read command line
        String hostName = HOST;
        int port = PORT;
        int selector = 7;
        try {
            System.out.println("-h(ost) host-name -p(ort) port-number -"
                + "s(elector) property-value-from-0-to-9");
            for (int i = 0; i < args.length; i++) {
               if (args[i].startsWith("-h") || args[i].startsWith("-host"))
```

```
                    hostName = args[++i];
                else if (args[i].startsWith("-p") || args[i].startsWith("-
port"))
                    port = Integer.parseInt(args[++i]);
                else if (args[i].startsWith("-s")
                    || args[i].startsWith("-selector"))
                    selector = Integer.parseInt(args[++i]);
            }
        } catch (Exception e) {
            System.out.println("Error in arguments");
            System.exit(1);
        }

        TopicConnection conn = null;
        try {
            // Create connection, session, etc.
            TopicConnectionFactory factory = new STCTopicConnectionFactory(
                hostName, port);
            conn = factory.createTopicConnection();
            TopicSession topicSession = conn.createTopicSession(true,
                Session.AUTO_ACKNOWLEDGE);
            Topic topic = topicSession.createTopic(TOPIC_NAME);

            // create subscriber
            String selectorString = PROP_NAME + " = '" + selector + "'";
            TopicSubscriber subscriber =
topicSession.createDurableSubscriber(
                topic,
                "SelectorSubscriber" + selector, selectorString, false);
            System.out.println("selector: " +
subscriber.getMessageSelector());

            // receive messages
            conn.start();
            for (Message msg = subscriber.receive(); msg != null;
                msg = subscriber.receive(1000)) {
               System.out.println("... Received 1 message with " + PROP_NAME
                    + " = " + msg.getStringProperty(PROP_NAME));
                topicSession.commit();
            }
            subscriber.close();
            topicSession.unsubscribe("SelectorSubscriber" + selector);
        } catch (Exception ex) {
            ex.printStackTrace();
        } finally {
            if (conn != null) {
                try {
                    conn.close();
                } catch (Exception ignore) {
                }
            }
        }
    }
}
```

COM VB Message Selector

```
Option Explicit

Dim MessageSelector As String
Dim topicConnectionFactory As New topicConnectionFactory
Dim topicConnection As topicConnection
Dim topicSession As topicSession
Dim topic As topic
```

```
Dim Publisher As TopicPublisher
Dim subscriber As TopicSubscriber
Dim MessagePublished  As TextMessage
Dim MessageReceived As TextMessage

Private Sub Form_Load()
' You should replace the host name and port number with the actual values
    topicConnectionFactory.HostName = "localhost"
    topicConnectionFactory.Port = 24053
' Create a topic connection
    Set topicConnection = topicConnectionFactory.CreateTopicConnection()
' Create a session
    Set topicSession = topicConnection.CreateTopicSession(True,
msAutoAcknowledge)
' Start the session
    topicConnection.Start
' Create a topic
    Set topic = topicSession.CreateTopic(txtTopicName)
' Create a publisher
    Set Publisher = topicSession.CreatePublisher(topic)
' Set message selector
    MessageSelector = "Name = 'John'"
' Create a subscriber with the message selector
    Set subscriber = topicSession.CreateSubscriber(topic, MessageSelector)
End Sub

Private Sub cmdPublish_Click()
' Create a text message
    Set MessagePublished =
topicSession.CreateTextMessage(txtPublished.Text)
    If chkMessageSelector Then
    ' Set the corresponding user property in the message, and subscriber
    ' should receive this message because it matches the message selector
        MessagePublished.SetProperty "Name", "John"
    End If
    ' Otherwise, don't set the user property in this message, and subscriber
    ' should not receive this message
' Publish this message
    Publisher.Publish MessagePublished
' Commit this message
    topicSession.Commit
End Sub

Private Sub cmdReceive_Click()
' Receive the message
    Set MessageReceived = subscriber.ReceiveNoWait
    If MessageReceived Is Nothing Then
        txtReceived = "No Message Received"
    Else
    ' Commit this message
        topicSession.Commit
        txtReceived = MessageReceived.Text
    End If

End Sub
```

4.2.10. XA Sample

XA compliance is achieved when cooperating software systems contain sufficient logic to ensure that the transfer of a single unit of data between those systems is neither lost nor duplicated because of a failure condition in one or more of the cooperating systems.

For more information on XA, see the *eGate Integrator User's Guide*.

Java XA Publisher

```
import java.io.InputStreamReader;
import javax.jms.*;
import javax.transaction.xa.*;
import com.stc.jms.client.STCXATopicConnectionFactory;
import com.stc.jms.util.XidImpl;
import java.io.IOException;

class XAPublisher {
    private static char readFromConsole(InputStreamReader console,
        String allowedCharacters) throws IOException {
        for (; ; ) {
            char c = (char) console.read();
            c = Character.toUpperCase(c);
            if (allowedCharacters.indexOf(c) >= 0) {
                return c;
            }
        }
    }

    public static void main(String args[]) {
        // Read command line
        String hostName = "localhost";
        int port = 18008;
        try {
            System.out.println("-h(ost) host-name -p(ort) port-number");
            for (int i = 0; i < args.length; i++) {
               if (args[i].startsWith("-h") || args[i].startsWith("-host"))
                    hostName = args[++i];
                else if (args[i].startsWith("-p") || args[i].startsWith("-
port"))
                    port = Integer.parseInt(args[++i]);
            }
        } catch (Exception e) {
            System.out.println("Error in arguments");
            System.exit(1);
        }


        String topicName = "XAPubSubSample";
        XATopicConnection topicConnection = null;

        System.out.println("pub topic name is " + topicName);
        try {
            // Create connection, session, etc.
            XATopicConnectionFactory tcf = new
STCXATopicConnectionFactory(hostName, port);
            topicConnection = tcf.createXATopicConnection();
            topicConnection.start();
            XATopicSession xaTopicSession =
topicConnection.createXATopicSession();
            XAResource resource = xaTopicSession.getXAResource();
            TopicSession topicSession = xaTopicSession.getTopicSession();
            Topic topic = topicSession.createTopic(topicName);
            TopicPublisher topicPublisher =
topicSession.createPublisher(topic);

            // Start the transaction
            System.out.println("... XAResource start");
            Xid xid = new XidImpl();
            resource.start(xid, XAResource.TMNOFLAGS);
```

```
            // Publish a msg
            TextMessage message = topicSession.createTextMessage("Hello
world");
            System.out.println("... Publishing message: " +
message.getText());
            topicPublisher.publish(message);

            // End the transaction and prepare to commit or rollback
            System.out.println("... XAResource end");
            resource.end(xid, XAResource.TMSUCCESS);
            System.out.println("XAResource prepare ....");
            resource.prepare(xid);

            // Ask user to commit or rollback
            boolean mustRollback = false;
            System.out.println("C or c to commit and R or r to rollback");
            InputStreamReader inputStreamReader = new
InputStreamReader(System.in);
            char answer = readFromConsole(inputStreamReader, "CR");
            if (answer == 'C') {
                System.out.println("... XAResource commit");
            } else if (answer == 'r' || answer == 'R') {
                System.out.println("... XAResource rollback");
                mustRollback = true;
            }

            // Commit or rollback
            if (mustRollback) {
                resource.rollback(xid);
            } else {
                resource.commit(xid, false);
            }
        } catch (Exception e) {
            System.out.println("Exception occurred: " + e.getMessage());
            e.printStackTrace();
        } finally {
            if (topicConnection != null) {
                try {
                    System.out.println("... Closing connection");
                    topicConnection.close();
                } catch (JMSException ignore) {
                }
            }
        }
    }
}
```

Java XA Subscriber

```
import java.io.*;
import javax.jms.*;
import javax.transaction.xa.*;
import com.stc.jms.util.XidImpl;
import com.stc.jms.client.STCXATopicConnectionFactory;

public class XASubscriber {
    private static char readFromConsole(InputStreamReader console,
        String allowedCharacters) throws IOException {
        for (; ; ) {
            char c = (char) console.read();
            c = Character.toUpperCase(c);
            if (allowedCharacters.indexOf(c) >= 0) {
                return c;
```

```
            }
        }
    }

    public static void main(String args[]) {
        // Get command line
        String hostName = "localhost";
        int port = 18008;
        try {
            System.out.println("-h(ost) host-name -p(ort) port-number");
            for (int i = 0; i < args.length; i++) {
               if (args[i].startsWith("-h") || args[i].startsWith("-host"))
                    hostName = args[++i];
                else if (args[i].startsWith("-p") || args[i].startsWith("-
port"))
                    port = Integer.parseInt(args[++i]);
            }
        } catch (Exception e) {
            System.out.println("Error in arguments");
            System.exit(1);
        }

        String topicName = "eGateXAPubSubSample";
        System.out.println("Topic name is " + topicName);

        XATopicConnection topicConnection = null;
        try {
            // Create connection, session, etc.
            XATopicConnectionFactory tcf = new
STCXATopicConnectionFactory(hostName, port);
            topicConnection = tcf.createXATopicConnection();
            topicConnection.start();
            XATopicSession xaTopicSession =
topicConnection.createXATopicSession();
            XAResource resource = xaTopicSession.getXAResource();
            Topic topic =
xaTopicSession.getTopicSession().createTopic(topicName);
            System.out.println("... XAResource start");
            TopicSubscriber topicSubscriber =
xaTopicSession.getTopicSession().createSubscriber(topic);

            InputStreamReader console = new InputStreamReader(System.in);
            for (;;) {
                // Start transaction
                System.out.println("Starting transaction...");
                Xid xid = new XidImpl();
                resource.start(xid, XAResource.TMNOFLAGS);

                // Receive msg
                System.out.println("Receiving msg...");
                Message message = topicSubscriber.receive();
                if (message instanceof TextMessage) {
                    System.out.println("... Reading message: " +
                        ((TextMessage) message).getText());
                } else {
                    System.out.println("Message of wrong type: " +
                        message.getClass().getName());
                }

                // Ask user for commit or rollback
                System.out.println(
                   "C or c to commit and R or r to rollback after prepare");
                char answer = readFromConsole(console, "CR");
```

```
                    // End transaction; prepare to commit or rollback
                    System.out.println("... XAResource end");
                    resource.end(xid, XAResource.TMSUCCESS);
                    System.out.println("... XAResource prepare");
                    resource.prepare(xid);

                    // Now commit or rollback
                    if (answer == 'C') {
                        System.out.println("... XAResource commit");
                        resource.commit(xid, false);
                    } else {
                        System.out.println("... XAResource rollback");
                        resource.rollback(xid);
                    }

                    // Allow user to stop
                    System.out.println(
                        "To end program, enter Q or q, then <return> or C to
receive again");
                    answer = readFromConsole(console, "QC");
                    if (answer == 'Q') {
                        break;
                    }
                }
            } catch (Exception e) {
                System.out.println("Exception occurred: " + e.toString());
            } finally {
                if (topicConnection != null) {
                    try {
                        System.out.println("... Closing connection");
                        topicConnection.close();
                    } catch (Exception ignore) {}
                }
            }
        }
    }
```

COM VB XA Sample

```
Option Explicit
Dim TopicObj As TopicTask
Dim QueueObj As QueueTask

Private Sub cmdPublish_Click()
    If chkTopic Then
        PublishTopic
    Else
        SendQueue
    End If
End Sub

Private Sub cmdReceive_Click()
    If chkTopic Then
        ReceiveTopic
    Else
        ReceiveQueue
    End If
End Sub

Private Sub PublishTopic()
    Set TopicObj = New TopicTask
    TopicObj.Send txtDestination, txtPublished, chkCommit
End Sub
```

```
Private Sub ReceiveTopic()
    Dim msg As String
    Set TopicObj = New TopicTask
    TopicObj.Receive txtDestination, msg, chkCommit
    If chkCommit Then
        txtReceived = msg
    Else
        txtReceived = "Aborted"
    End If
End Sub

Private Sub SendQueue()
    Set QueueObj = New QueueTask
    QueueObj.Send txtDestination, txtPublished, chkCommit
End Sub

Private Sub ReceiveQueue()
    Dim msg As String
    Set QueueObj = New QueueTask
    QueueObj.Receive txtDestination, msg, chkCommit
    If chkCommit Then
        txtReceived = msg
    Else
        txtReceived = "Aborted"
    End If
End Sub
```

4.2.11. The Compensating Resource Manager

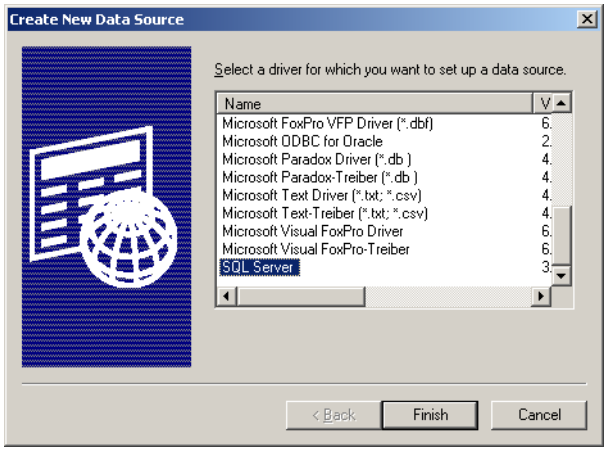
Creating a SQL Server Database

The samples provided are designed using an SQL Server Database.

1. Create a SQL Server database, using the name “CRM” for the purpose of testing the samples.
2. Create a table, using the name “Messages”.
3. Create two columns in the table, “UID” and “Message”.

4 From Settings->ControlPanel->AdministrativeTools->DataSources, Add an SQL Database source.

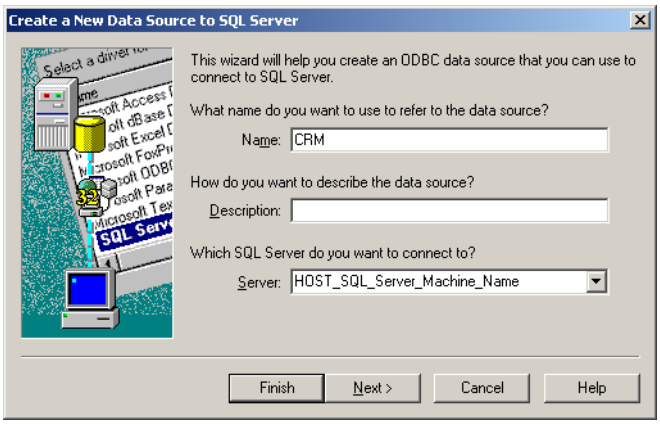
Figure 21 SQL Database Source



5 Provide the name of the data source, a description if desired, and the machine name on which SQL Server is running.

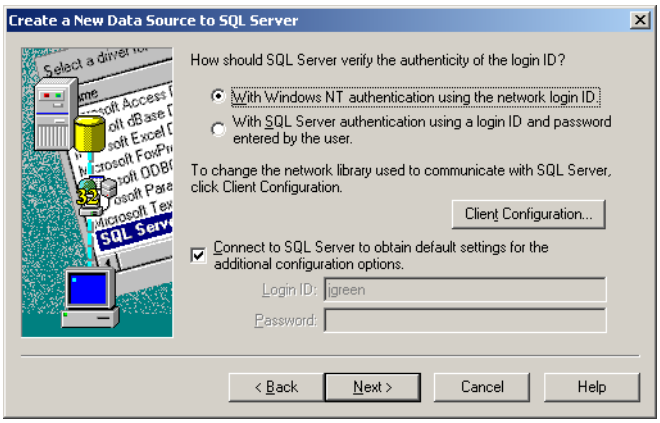
Important: *You will not be able to continue until a successful connection is made.*

Figure 22 SQL Datasource



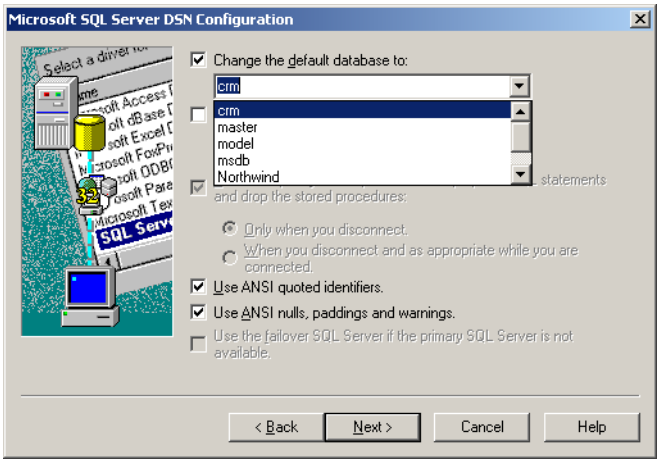
6 Ensure that the authentication and login settings correspond to Figure 23 below.

Figure 23 Login Settings



7 Select the recently created database as the default from the drop-down list, as shown in Figure 24 below. Click Next to continue.

Figure 24 Default SQL Server Database



8 Click Finish.

Configuring the Compensating Resource Manager (CRM)

When planning your CRM implementation, you cannot assume that the same instance of the CRM Compensator that processes the set of method calls in the prepare phase will process the method calls in the commit phase. If one of the clients attempts to commit a transaction, and someone inadvertently disconnects the power source during the commit phase, the prepare method calls will not be repeated during recovery, and the Compensator receives a set of abort or commit method calls.

Both the CRM Worker and Compensator are COM+ components and they must be configured using the Windows 2000 Component Services Explorer function properly. The CRM Worker and CRM Compensator must be installed into the same COM+ application that was completed above. (See **"Configuring the Compensating Resource Manager (CRM)" on page 68** for more information.)

Note: *You must create the "CRM" database before attempting to use any sample code. See* **"Creating a SQL Server Database" on page 66** *for more information.*

For this section, two sample files will be used, the samples can be found on the Installation CD under Samples\jmsapi\com

CRMTest.vdb

CRMTest.dll

1. Open CRMTest.vdp. Four files open in the project. Follow the comments in the code to modify the sample to your system requirements. The comments appear in **bold** typeface in the code samples that follow.

InsertMessage.cls

```
Option Explicit

Sub Add(message As String, commit As Boolean)
    On Error GoTo errHandler
    Dim ObjCtx As ObjectContext
    Set ObjCtx = GetObjectContext()
    If ObjCtx Is Nothing Then
        MsgBox "Application is not running in COM+"
        Exit Sub
    End If

' Before start this CRM sample dll, you should create a database
' called "crm", and create a table named "Messages" with two
' columns, one column is "ID", and the other one is "Message"

' You can replace the following steps to use another resource manager
' i.e., the Oracle DBMS

    Dim adoPrimaryRS As Recordset
    Dim db As ADODB.Connection
    Set db = New ADODB.Connection
    db.CursorLocation = adUseClient
' Create a data source name
    db.Open "PROVIDER=MSDASQL;dsn=CRM;uid=sa;pwd=sa;"
    Set adoPrimaryRS = New ADODB.Recordset
    adoPrimaryRS.Open "select Message from Messages", db, adOpenStatic,
         adLockOptimistic
    adoPrimaryRS.AddNew "Message", message
    db.Close
    Set db = Nothing
    Set adoPrimaryRS = Nothing
     If Not commit Then
        ObjCtx.SetAbort
        Err.Raise vbObjectError + 1024, , "Instruction to Abort the
         Transaction"
    End If
    Exit Sub
errHandler:
    ObjCtx.SetAbort
    Err.Raise Err
End Sub
```

TwoTasks.cls

```
Option Explicit
' In this CRM sample, there are two tasks that must either both succeed
' or both abort
```

```
' The TopicTask is to send a message to the JMS server
Dim topicObj As TopicTask
' The InsertMessage task is to insert a message into a SQL table
Dim MessageObj As InsertMessage

Sub Send(topicName As String, msg As String, commit As Boolean)
    On Error GoTo errHandler
    Dim ObjCtx As ObjectContext
    Set ObjCtx = GetObjectContext()

    Set topicObj = New TopicTask
    Set MessageObj = New InsertMessage
' First task is to send a message to the JMS server
    topicObj.Send topicName, msg, True
' Second task is to add this message into the "Messages" table
    MessageObj.Add msg, True
    If Not commit Then
        ObjCtx.SetAbort
        Err.Raise vbObjectError + 1024, , "Instruction to Abort the
Transaction"
    End If
    Exit Sub
errHandler:
    ObjCtx.SetAbort
    Err.Raise Err
End Sub

Sub Receive(topicName As String, msg As String, commit As Boolean)
    On Error GoTo errHandler
    Dim ObjCtx As ObjectContext
    Set ObjCtx = GetObjectContext()
    If ObjCtx Is Nothing Then
        MsgBox "Application is not running in COM+"
        Exit Sub
    End If
    Set topicObj = New TopicTask
' Receive a message
    topicObj.Receive topicName, msg, True
    If Not commit Then
        ObjCtx.SetAbort
        Err.Raise vbObjectError + 1024, , "Instruction to Abort the
Transaction"
    End If
    Exit Sub
errHandler:
    ObjCtx.SetAbort
    Err.Raise Err
End Sub
```

TopicTask.cls

```
Option Explicit

Dim XATopicConnectionFactory As XATopicConnectionFactory
Dim XATopicConnection As XATopicConnection
Dim XATopicSession As XATopicSession
Dim TopicSession As TopicSession
Dim Topic As Topic
Dim subscriber As TopicSubscriber
Dim Publisher As TopicPublisher
Dim MessagePublished  As TextMessage
Dim MessageReceived As TextMessage

Sub Send(topicName As String, msg As String, commit As Boolean)
```

```
    On Error GoTo errHandler
    Dim ObjCtx As ObjectContext
    Set ObjCtx = GetObjectContext()
    If ObjCtx Is Nothing Then
        MsgBox "Application is not running in COM+"
        Exit Sub
    End If
' Create a XA topic connection factory
    Set XATopicConnectionFactory = New XATopicConnectionFactory
' You should replace the host name and port number with the actual values
    XATopicConnectionFactory.HostName = "localhost"
    XATopicConnectionFactory.Port = 24053
' Create a XA topic connection
    Set XATopicConnection =
XATopicConnectionFactory.CreateXATopicConnection()
' Create a XA topic session
    Set XATopicSession = XATopicConnection.CreateXATopicSession()
    Set TopicSession = XATopicSession.TopicSession
' Create a topic
    Set Topic = TopicSession.CreateTopic(topicName)
' Start the XA topic session
    XATopicConnection.Start
' Create a publisher
    Set Publisher = TopicSession.CreatePublisher(Topic)
' Create a text message
    Set MessagePublished = TopicSession.CreateTextMessage(msg)
' Publish the message
    Publisher.Publish MessagePublished
    If Not commit Then
        ObjCtx.SetAbort
        Err.Raise vbObjectError + 1024, , "Instruction to Abort the
Transaction"
    End If
    Exit Sub
errHandler:
    ObjCtx.SetAbort
    Err.Raise Err
End Sub

Sub Receive(topicName As String, msg As String, commit As Boolean)
    On Error GoTo errHandler
    Dim ObjCtx As ObjectContext
    Set ObjCtx = GetObjectContext()
    If ObjCtx Is Nothing Then
        MsgBox "Application is not running in COM+"
        Exit Sub
    End If
' Create a XA topic connection factory
    Set XATopicConnectionFactory = New XATopicConnectionFactory
' You should replace the host name and port number with the actual values
    XATopicConnectionFactory.HostName = "localhost"
    XATopicConnectionFactory.Port = 24053
' Create a XA topic connection
    Set XATopicConnection =
XATopicConnectionFactory.CreateXATopicConnection()
' Create a XA topic session
    Set XATopicSession = XATopicConnection.CreateXATopicSession()
    Set TopicSession = XATopicSession.TopicSession
' Create a topic
    Set Topic = TopicSession.CreateTopic(topicName)
' Start the XA topic session
    XATopicConnection.Start
' Create a subscriber
```

```
    Set subscriber = TopicSession.CreateDurableSubscriber(Topic,
"TopicSubscriber")
' receive a message
    Set MessageReceived = subscriber.ReceiveNoWait
    If MessageReceived Is Nothing Then
        msg = "No Message Received"
    Else
        msg = MessageReceived.Text
    End If
    If Not commit Then
        ObjCtx.SetAbort
        Err.Raise vbObjectError + 1024, , "Instruction to Abort the
Transaction"
    End If
    Exit Sub
errHandler:
    ObjCtx.SetAbort
    Err.Raise Err
End Sub
```

QueueTasks.cls

```
Option Explicit

Dim XAQueueConnectionFactory As XAQueueConnectionFactory
Dim XAQueueConnection As XAQueueConnection
Dim XAQueueSession As XAQueueSession
Dim QueueSession As QueueSession
Dim Queue As Queue
Dim QueueReceiver As QueueReceiver
Dim QueueSender As QueueSender
Dim MessagePublished  As TextMessage
Dim MessageReceived As TextMessage

Sub Send(QueueName As String, msg As String, commit As Boolean)
    On Error GoTo errHandler
    Dim ObjCtx As ObjectContext
    Set ObjCtx = GetObjectContext()
    If ObjCtx Is Nothing Then
        MsgBox "Application is not running in COM+"
        Exit Sub
    End If
' Create a XA queue connection factory
    Set XAQueueConnectionFactory = New XAQueueConnectionFactory
' You should replace the host name and port number with the actual values
    XAQueueConnectionFactory.HostName = "localhost"
    XAQueueConnectionFactory.Port = 24053
' Create a XA queue connection
    Set XAQueueConnection =
XAQueueConnectionFactory.CreateXAQueueConnection()
' Create a XA queue session
    Set XAQueueSession = XAQueueConnection.CreateXAQueueSession()
    Set QueueSession = XAQueueSession.QueueSession
' Create a queue
    Set Queue = QueueSession.CreateQueue(QueueName)
' Start the XA queue session
    XAQueueConnection.Start
' Create a queue sender
    Set QueueSender = QueueSession.CreateSender(Queue)
' Create a text message
    Set MessagePublished = QueueSession.CreateTextMessage(msg)
' Send a message
    QueueSender.Send MessagePublished
    If Not commit Then
```

```
        ObjCtx.SetAbort
        Err.Raise vbObjectError + 1024, , "Instruction to Abort the
Transaction"
    End If
    Exit Sub
errHandler:
    ObjCtx.SetAbort
    Err.Raise Err
End Sub

Sub Receive(QueueName As String, msg As String, commit As Boolean)
    On Error GoTo errHandler
    Dim ObjCtx As ObjectContext
    Set ObjCtx = GetObjectContext()
    If ObjCtx Is Nothing Then
        MsgBox "Application is not running in COM+"
        Exit Sub
    End If
' Create a XA Queue connection factory
    Set XAQueueConnectionFactory = New XAQueueConnectionFactory
' You should replace the host name and port number with the actual values
    XAQueueConnectionFactory.HostName = "localhost"
    XAQueueConnectionFactory.Port = 24053
' Create a XA queue connection
    Set XAQueueConnection =
XAQueueConnectionFactory.CreateXAQueueConnection()
' Create a XA queue session
    Set XAQueueSession = XAQueueConnection.CreateXAQueueSession()
    Set QueueSession = XAQueueSession.QueueSession
' Create a queue
    Set Queue = QueueSession.CreateQueue(QueueName)
' Start the XA queue session
    XAQueueConnection.Start
' Create a queue receiver
    Set QueueReceiver = QueueSession.CreateReceiver(Queue)
' Receive a message
    Set MessageReceived = QueueReceiver.ReceiveNoWait
    If MessageReceived Is Nothing Then
        msg = "No Message Received"
    Else
        msg = MessageReceived.Text
    End If
     If Not commit Then
        ObjCtx.SetAbort
        Err.Raise vbObjectError + 1024, , "Instruction to Abort the
Transaction"
    End If
    Exit Sub
errHandler:
    ObjCtx.SetAbort
    Err.Raise Err
End Sub
```

2 Copy **client.exe** (located in the CRM sample folder) to the machine upon which the external code is to run.

3 Register **CRMTest.dll** by performing the following:

From the command prompt of the external system, register the file **CRMTest.dll** into the Windows 2000 registry by doing the following:

```
regsvr32 your_path_location\stc_mscom.dll
```

4 From the following location:

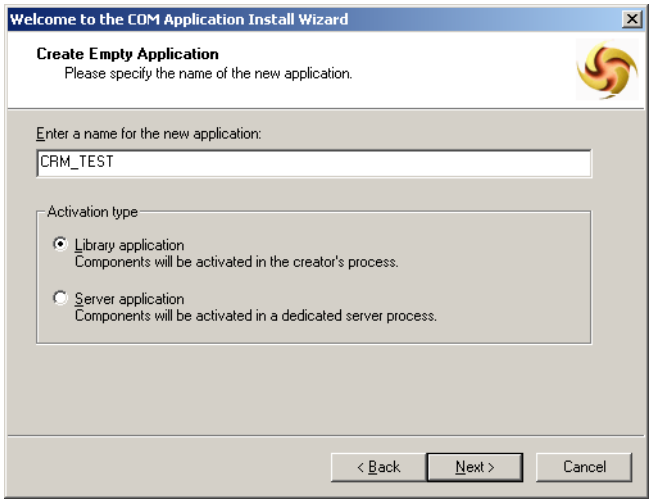
`Settings->Control Panel->Administrative Tools->Component Services`

Expand the Component Services folder. Right click on Com+ Applications.

Select New->Application. The COM Application Install Wizard opens. Click Next to continue.

5 Enter a CRM_TEST as the name of the new application. (Any name could be used.) Select Library application as the Application Type.

Figure 25 CRM_TEST Application



6 Click Next to continue, and then click Finish.

7 Expand the CRM_TEST component, right- click the Components folder, and then click New Component. The COM Component Install Wizard opens. Click Next to continue.

8 Click Install New Component(s). Browse to the location of the recently compiled CRMTest.dll. Click Open. Accept the remainder of the default settings.

4.3 Implementing Secure Socket Layers (SSL)

A TCP/IP connection between a client system and a server can be secured by implementing Secure Socket Layers (SSL). The SeeBeyond Message Service listens on two ports at the same time. While the default port listens for non-SSL connections, the other can be restricted to only accept an SSL connection.

Connecting to the Message Server Using SSL:

The following sample shows how to connect to the message server using SSL by instantiating a connection factory directly.

```
import com.stc.jms.client.STCQueueConnectionFactory;
import javax.jms.*;
import java.util.*;

public class Sample {
```

```
  public static void main(String[] args) {
    try {
      Properties p = new Properties();
      p.put("com.stc.jms.sockets.ServerPort", "9225");
      p.put("com.stc.jms.sockets.ServerHost", "localhost");
      p.put("com.stc.jms.ssl.authenticationmode", "TrustAll");
      QueueConnectionFactory fact
        = new STCQueueConnectionFactory(p);
      QueueConnection conn = fact.createQueueConnection();
      QueueSession session = conn.createQueueSession(false,
        Session.AUTO_ACKNOWLEDGE);
      Queue dest = session.createQueue("Test");
      QueueSender producer = session.createSender(dest);
      TextMessage msg1 = session.createTextMessage("Hello world!");
      producer.send(msg1, DeliveryMode.NON_PERSISTENT,
        Message.DEFAULT_DELIVERY_MODE, 0);
      conn.close();
    } catch (JMSException ex) {
      ex.printStackTrace();
    }
  }
}
```

4.4 Logging

The Client Runtime will attempt to load the **log4j** logging system unless the environment variable `com.stc.jms.nologging` is set to "**True**". The **log4j** logging system is provided by Apache. For for additional information see **www.apache.org**. The **log4j** file is only required when connecting to JNDI on the Integration Server.

To configure **log4j**, a properties file can be used. A sample **log4j.properties** file is part of the Kit and is included.

4.5 Multi Threading Apartment (MTA) Support

The COM components support the multi threaded apartment model (MTA). Multiple threads in the application can use the COM component at the same time, e.g. multiple threads can create sessions, send messages and wait for messages to be received. It should be noted that the multi-threaded model follows the multi-threaded programming model outlined in the Sun JMS spec 1.0.2. In essence, this defines the session and its associated objects as single threaded contexts.

Chapter 5

Client Libraries for the SeeBeyond COM+ APIs

This chapter provides a detailed discussion of the Application Program Interfaces (APIs) for SeeBeyond JMS. It is divided by language into the following sections:

For COM+

- **The COM+ API for the SeeBeyond Message Service** on page 76

The SeeBeyond Message Service includes additional API support in the following languages:

For Java

- **The JMS API for the SeeBeyond Message Service** on page 142

For C

- **The C API** on page 173
- **Error Codes and Messages in the C API** on page 273

For C++

The C++ API for the SeeBeyond Message Service on page 299

5.1 The COM+ API for the SeeBeyond Message Service

5.1.1. Supported Java Message Service (JMS) Classes for COM+

eGate supports the following list of the Java Message Service (JMS) COM+ APIs. If you need additional information for each of the classes and methods, please refer to Sun Microsystems web site at:

http://java.sun.com/products/jms/javadoc-102a/javax/jms/package-summary.html

You may also find useful the following books:

- *Java Message Service*, O'Reilly, December 2000, ISBN: 0596000685
- *Professional JMS*, Wrox Press, March 2001, ISBN: 1861004931
- *Professional Java Server Programming - J2EE Edition*, Wrox Press, September 2000, ISBN: 1861004656

5.1.2. The BytesMessage Object

A **BytesMessage** is used to send a message containing a stream of uninterrupted bytes. It inherits Message and adds a bytes message body. The receiver of the message supplies the interpretation of the bytes. Member of the Message Object.

The Acknowledge Method

Acknowledges the receipt of current and previous messages.

```
BytesMessage.acknowledge
```

The ClearBody Method

Clears the body of a message, leaving the message header values and property entries intact.

```
BytesMessage.ClearBody
```

The ClearProperties Method

Clears the properties from a message, leaving the message header fields and body intact.

```
BytesMessage.ClearProperties
```

The GetProperty Method

Returns the Visual Basic data type property value with the given name, into the Message.

```
BytesMessage.GetProperty(name As String)
```

Name	Description
name	The name of the property.

The PropertyExists Method

Checks whether a value for a specific property exists.

```
BytesMessage.PropertyExists(name as String)
```

Name	Description
name	The name of the property to check.

The ReadBoolean Method

Reads a Boolean value from the bytes message stream.

```
BytesMessage.ReadBoolean() As Boolean
```

The ReadByte Method

Reads a signed 8-bit value from the bytes message stream.

```
BytesMessage.ReadByte()As Byte
```

The ReadBytes Method

Reads a portion of the bytes message stream.

```
BytesMessage.ReadBytes(value, [length])As Long
```

Name	Description
value	The buffer the data is read into.
length	The number of bytes read.

The ReadChar Method

Reads a Unicode character value from the bytes message stream.

```
BytesMessage.ReadChar() As Integer
```

The ReadDouble Method

Reads a double from the bytes message stream.

```
BytesMessage.ReadDouble() As Double
```

The ReadFloat Method

Reads a float from the bytes message stream.

```
BytesMessage.ReadFloat() As Single
```

The ReadInt Method

Reads a signed 32-bit integer from the bytes message stream.

```
BytesMessage.ReadInt() As Long
```

The ReadLong Method

Reads a signed 64-bit integer from the bytes message stream.

```
BytesMessage.ReadLong() As Currency
```

The ReadShort Method

Reads a signed 16-bit number from the bytes message stream.

```
BytesMessage.ReadShort() As Integer
```

The ReadUnsignedByte Method

Reads an unsigned 8-bit number from the bytes message stream.

```
BytesMessage.ReadUnsignedByte() As Long
```

The ReadUnsignedShort Method

Reads an unsigned 16-bit number from the bytes message stream

```
BytesMessage.ReadUnsignedShort() As Long
```

The ReadUTF Method

ReadUTF reads the string that was encoded using a modified UTF-8 format from the bytes message stream.

```
BytesMessage.ReadUTF() As String
```

The Reset Method

The Reset method puts the message body in read-only mode, and repositions the stream of bytes to the beginning.

```
BytesMessage.Reset
```

The SetProperty Method

The SetProperty method sets a Visual Basic data type property value with the given name, into the Message.

```
BytesMessage.SetProperty(name As String, value)
```

Name	Description
name	Name of the property.
value	Value to set.

The WriteBoolean Method

WriteBoolean writes to the bytes message stream as a 1-byte value.

```
BytesMessage.WriteBoolean(value as Boolean)
```

Name	Description
value	Write a boolean to the bytes message stream as a 1 byte value. The value true is written out as the value (byte)1; the value false is written out as the value (byte)0.

The WriteByte Method

WriteByte writes to the bytes message stream as a 1-byte value

```
BytesMessage.WriteByte(value As Byte)
```

Name	Description
value	The byte value to be written

The WriteBytes Method

WriteBytes writes a byte array, or a portion of the byte array, to the bytes message stream

```
BytesMessage.WriteBytes(value, [offset], [length])
```

Name	Description
value	The byte array value to be written.
offset	The initial offset within the byte array.
length	The number of bytes to use.

The WriteChar Method

WriteChar writes a char to the bytes message stream as a 2-byte value, high byte first

```
BytesMessage.WriteChar(value As integer)
```

Name	Description
value	The Char value to be written.

The WriteDouble Method

Convert the double parameter value to a long, and then writes an 8-byte long value to the bytes message stream (high byte is written first).

```
BytesMessage.WriteDouble(value As Double)
```

Name	Description
value	The double value to write to the message stream.

The WriteFloat Method

Convert the float argument to an long, and then writes that long value to the bytes message stream as a 4-byte quantity, high byte first

```
BytesMessage.WriteFloat(Value As Single)
```

Name	Description
value	The float value to be written.

The WriteInt Method

Write an int to the bytes message stream as four bytes, high byte first.

```
BytesMessage.WriteInt(value As Long)
```

Name	Description
value	The float value to be written.

The WriteLong Method

WriteLong writes a long to the bytes message stream as eight bytes, high byte first

```
BytesMessage.WriteLong(value As Currency)
```

Name	Description
value	The WriteLong is written as a currency.

The WriteObject Method

Currently not supported

The WriteShort Method

WriteShort writes a short to the bytes message stream as two bytes, high byte first

```
BytesMessage.WriteShort(value As Integer)
```

Name	Description
value	The short that is written.

The WriteUTF Method

WriteUTF writes a string to the bytes message stream using UTF-8 encoding in a machine-independent manner

```
BytesMessage.WriteUTF(value As String)
```

Name	Description
value	The `String` value that is written.

5.1.3. Properties of the BytesMessage Object

The CorrelationID Property

The CorrelationID property sets or returns correlation id values that are either JMS IQ Manager specific message ID's or application-specific strings.

```
BytesMessage.CorrelationID = String
String = BytesMessage.CorrelationID
```

The CorrelationIDAsBytes Property

Currently not supported.

The DeliveryMode Property

The DeliveryMode property sets or returns the delivery mode for this message as either msNonPersistent, or msPersistent. The default value is msDefaultDeliveryMode (msPersistent).

```
DeliveryMode = BytesMessageConstant
BytesMessageConstant = DeliveryMode
```

Name	Description
msPersistent	This mode instructs the JMS IQ Manager to log the message to storage as part of the client's send operation.
msNon_Persistent	This is the lowest overhead delivery mode because it does not require the message to be logged to storage. the JMS IQ Manager delivers a Non_Persistent message with a one-time-only delivery. If the JMS IQ Manager goes down, the message will be lost.

The Destination Property

Currently not supported.

The Expiration Property

The Expiration property sets or returns the message expiration time in milliseconds.

```
BytesMessage.Expiration = Currency
Currency = BytesMessage.Expiration
```

The MessageID Property

The MessageID property sets or returns the value of the uniquely assigned identifier in the message header.

```
BytesMessage.MessageID = String
String = BytesMessage.MessageID
```

The Priority Property

```
Currently not supported.
```

The Redelivered Property

The Redelivered property sets or returns an indication as to whether this message is redelivered.

```
BytesMessage.Redelivered = Boolean
Boolean = BytesMessage.Redelivered
```

The ReplyTo Property

The ReplyTo property sets or returns were a reply to this message will be sent. Destination can be a Topic, Queue, Temporary Topic, or a Temporary Queue.

```
BytesMessage.ReplyTo = Destination
Destination = BytesMessage.ReplyTo
```

The Timestamp Property

The TimeStamp property sets or returns the message timestamp.

```
BytesMessage.Timestamp = Currency
Currency = BytesMessage.Timestamp
```

The Type Property

The Type property sets or returns the message type.

```
BytesMessage.Type = String
String = BytesMessage.Type
```

5.1.4. The Connection Object

A Connection is a client's active connection to its provider. This is an abstract interface.

The Start Method

The Start method starts or restarts the delivery of a transaction connection's incoming messages.

```
Connection.Start
```

The Stop Method

The Stop methods temporarily stops the delivery of incoming messages from a transaction connection.

```
Connection.Stop
```

5.1.5. Properties of the Connection Object

The ClientID Property

ClientID sets or returns the client identifier for this connection. This value is JMS IQ Manager specific.

```
Connection.ClientID = String
String = Connection.ClientID
```

The MetaData Property

This property is not currently supported.

5.1.6. The ConnectionFactory Object

A ConnectionFactory encapsulates a set of connection configuration parameters that has been defined by your administrator. This is an abstract interface.

There are no methods currently associated with this object.

5.1.7. Properties of the ConnectionFactory Object

The HostName Property

HostName is a property that sets or returns the name of the host where Message Service is running.

```
ConnectionFactory.HostName = String
String = ConnectionFactory.HostName
```

The Port Property

The Port property sets or returns the port number at which the Message Service is listening, default value is 24053

```
ConnectionFactory.Port = Long
Long = ConnectionFactory.Port
```

The PortOffset Property

The PortOffset property sets or returns the port offset number of the Message Service if more then one Message Service is running on the same host machine and using the same port number

```
ConnectionFactory.PortOffset = Long
Long = ConnectionFactory.PortOffset
```

5.1.8. The Connection MetaData Object

This Object is currently not supported.

5.1.9. The MapMessage Object

The MapMessage is used to send a set of name-value pairs where names are Strings and values are primitive data types. Member of the Message Object.

The Acknowledge Method

Acknowledges the receipt of current and previous messages.

```
MapMessage.Acknowledge
```

The ClearBody Method

Clears the body of a message, leaving the message header values and property entries intact.

```
MapMessage.ClearBody
```

The ClearProperties Method

Clears the properties from a message, leaving the message header fields and body intact.

```
MapMessage.ClearProperties
```

The GetBoolean Method

The GetBoolean method returns the boolean value with the given name

```
MapMessage.GetBoolean() As Boolean
```

Name	Description
name	The name of the Boolean property.

The GetByte Method

The GetByte method returns the byte value with the given name.

```
MapMessage.GetByte(name as a String) As Byte
```

Name	Description
name	The name of the byte.

The GetBytes Methods

The GetBytes method returns the byte array value with the given name as a variable.

```
MapMessage.GetBytes(name As String, length As Long)
```

Name	Description
name	The name of the byte property.
length	The length of the property

The GetChar Method

The GetChar property returns the Unicode character value with the given name.

```
MapMessage.GetChar(name As String) As Integer
```

Name	Description
name	The name of the Unicode character.

The GetDouble Method

The GetDouble method returns the double value with the given name.

```
MapMessage.GetDouble(name As String) As Double
```

Name	Description
name	The name of the double property.

The GetFloat Method

The GetFloat method returns the float value with the given name.

```
MapMessage.GetFloat(name As String)
```

Name	Description
name	The name of the float property.

The GetInt Method

The GetInt method returns the long value with the given name

```
MapMessage.GetInt(name as a String) As Long
```

Name	Description
name	The name of the integer.

The GetLong Method

The GetLong method returns the currency value with the given name.

```
MapMessage.GetLong(name As String)As Currency
```

Name	Description
name	The name of the currency property.

The GetObject Method

The GetObject method is currently not supported.

The GetProperty Method

The GetProperty method returns the Visual Basic data type property value with the given name, into the Message.

```
MapMessage.GetProperty(name As String)
```

Name	Description
name	Name of the currency property.

The GetShort Method

The GetShort method returns the short value with the given name.

```
MapMessage.GetShort (name As String) As Integer
```

Name	Description
name	The name of the short currency property.

The GetString Method

Return the String value with the given name

```
MapMessage.GetString(name As String) As String
```

Name	Description
name	The name of the String property.

The ItemExists Method

The ItemExists method checks to verify if an item exists in the MapMessage.

```
MapMessage.ItemExists(name As String) As Boolean
```

Name	Description
name	The name of the item to check.

The PropertyExists Method

The PropertyExists method checks if a property value exists.

```
MapMessage.PropertyExists (name As String) As Boolean
```

Name	Description
name	The name of the property value.

The SetBoolean Method

The SetBoolean method sets a boolean property value with the given name, into the Message.

```
MapMessage.SetBoolean (name As String, value As Boolean)
```

Name	Description
name	The name of the property value.
value	The value to set in the message.

The SetByte Method

The SetByte method sets a byte value with the given name, into the Map.

```
MapMessage.SetByte(name As String, value As Byte)
```

Name	Description
name	The name of the byte property.
value	The byte property value to set in the message.

The SetBytes Method

The SetBytes method sets a byte array or a portion of value with the given name, into the Map.

```
MapMessage.SetBytes(name As String, value, [offset], [length])
```

Name	Description
name	The name of the Bytes property.
value	The byte array value to set in the Map.
offset	The initial offset within the byte array.
length	The number of bytes to use.

The SetChar Method

The SetChar method sets a Unicode character value with the given name, into the Map.

```
MapMessage.SetChar(name As String, value As Integer)
```

Name	Description
name	The name of the Unicode character.
value	The Unicode character value to set in the Map.

The SetDouble Method

The SetDouble method sets a double value with the given name, into the Map.

```
MapMessage.SetDouble(name As String, value As Double)
```

Name	Description
name	The name of the double property.
value	The double property value to set in the map.

The SetFloat Methods

The SetFloat method sets a float value with the given name, into the Map.

```
MapMessage.SetFloat(name As String, value As Single)
```

Name	Description
name	The name of the float property.
value	The the float value to set in the map.

The SetInt Method

Set an long value with the given name, into the Map

```
MapMessage.SetInt(name As String, value As Long)
```

Name	Description
name	The name of the long property.
value	The long property value to set in the message.

The SetLong Method

The SetLong method sets a currency value with the given name, into the Map.

```
MapMessage.SetLong(name As String, value As Currency)
```

Name	Description
name	The name of the currency property.
value	The currency property value to set in the message.

The SetObject Method

This method is currently not supported.

The SetProperty Method

Sets a Visual Basic data type property value with the given name, into the Message.

```
MapMessage.SetProperty(name As String, value)
```

Name	Description
name	The name of the property.
value	The value to set.

The SetShort Method

The SetShort method sets a short value with the given name, into the Map.

```
MapMessage.SetShort(name As String, value As Integer)
```

Name	Description
name	The name of the short property.
value	The integer property value to set in the map.

The SetString Method

The SetString method sets a String value with the given name, into the Map.

```
MapMessage.SetString(name As String, value As String)
```

Name	Description
name	The name of the string property.
value	The string value to set into the map.

5.1.10. Properties of the MapMessage Object

The CorrelationID Property

The CorrelationID property sets or returns correlation id values that are either JMS IQ Manager specific message ID's or application-specific strings.

```
Mapessage.CorrelationID = String
String = MapMessage.CorrelationID
```

The CorrelationIDAsBytes Property

Currently not supported.

The DeliveryMode Property

The DeliveryMode property sets or returns the delivery mode for this message as either msNonPersistent, or msPersistent. The default value is msDefaultDeliveryMode (msPersistent).

```
DeliveryMode= DeliveryModeConstant
DeliveryModeConstant = DeliveryMode
```

Name	Description
msPersistent	This mode instructs the JMS IQ Manager to log the message to storage as part of the client's send operation.
msNon_Persistent	This is the lowest overhead delivery mode because it does not require the message to be logged to storage. the JMS IQ Manager delivers a Non_Persistent message with a one-time-only delivery. If the JMS IQ Manager goes down, the message will be lost.

The Destination Property

Currently not supported.

The Expiration Property

The Expiration property sets or returns the message expiration time in milliseconds.

```
MapMessage.Expiration = Currency
Currency = MapMessage.Expiration
```

The MapNames Property

The MapNames property returns the Map message's names as an array of String. (read-only)

```
MapMessage.MapNames = Variant
Variant = MapMessage.MapNames
```

The MessageID Property

The MessageID property sets or returns the value of the uniquely assigned identifier in the message header.

```
MapMessage.MessageID = String
String = MapMessage.MessageID
```

The Priority Property

Currently not supported.

The Redelivered Property

The Redelivered property sets or returns an indication as to whether this message is redelivered.

```
MapMessage.Redelivered = Boolean
Boolean = MapMessage.Redelivered
```

The ReplyTo Property

The ReplyTo property sets or returns were a reply to this message will be sent. Destination object could be a Topic, Queue, TemporaryTopic, or a TemporaryQueue.

```
MapMessage.ReplyTo = Destination
Destination = MapMessage.ReplyTo
```

The Timestamp Property

The TimeStamp property sets or returns the message timestamp.

```
MapMessage.Timestamp = Currency
Currency = MapMessage.Timestamp
```

The Type Property

The Type property sets or returns the message type.

```
MapMessage.Type = String
String = MapMessage.Type
```

5.1.11. The Message Object

The Message interface is the root interface of all JMS messages. It defines the JMS header and the acknowledge method used for all messages.

Subclasses of the Message Object include: BytesMessage, MapMessage, TextMessage, and StreamMessage.

The Acknowledge Method

Acknowledges the receipt of current and previous messages.

```
Message.acknowledge
```

The ClearBody Method

Clears the body of a message, leaving the message header values and property entries intact.

```
Message.ClearBody
```

The ClearProperties Method

Clears the properties from a message, leaving the message header fields and body intact.

```
Message.ClearProperties
```

The GetProperty Method

Returns the Visual Basic data type property value with the given name, into the Message.

```
Message.GetProperty(name As String)
```

Name	Description
name	The name of the property.

The PropertyExists Method

Checks whether a value for a specific property exists.

```
Message.PropertyExists(name) As Boolean
```

Name	Description
name	The name of the property to check.

The SetProperty Method

The SetProperty method sets a Visual Basic data type property value with the given name, into the Message.

```
Message.SetProperty(name As String, value)
```

Name	Description
name	The name of the byte property.
value	The value to set.

5.1.12. Properties of the Message Object

The CorrelationID Property

The CorrelationID property sets or returns correlation id values that are either JMS IQ Manager specific message ID's or application-specific strings.

```
Message.CorrelationID = String
String = Message.CorrelationID
```

The CorrelationIDAsBytes Property

The CorrelationIDAsBytes is not currently supported.

The DeliveryMode Property

The DeliveryMode property sets or returns the delivery mode for this message as either msNonPersistent, or msPersistent. The default value is msDefaultDeliveryMode (msPersistent).

```
DeliveryMode = MessageConstant
MessageConstant = DeliveryMode
```

Name	Description
msPersistent	This mode instructs the JMS IQ Manager to log the message to storage as part of the client's send operation.
msNon_Persistent	This is the lowest overhead delivery mode because it does not require the message to be logged to storage. the JMS IQ Manager delivers a Non_Persistent message with a one-time-only delivery. If the JMS IQ Manager goes down, the message will be lost.

The Destination Property

Currently not supported.

The Expiration Property

The Expiration property sets or returns the message expiration time in milliseconds.

```
Message.Expiration = Currency
Currency = Message.Expiration
```

The MessageID Property

The MessageID property sets or returns the value of the uniquely assigned identifier in the message header.

```
Message.MessageID = String
String = Message.MessageID
```

The Priority Property

Currently not supported.

The Redelivered Property

The Redelivered property sets or returns an indication as to whether this message is redelivered.

```
Message.Redelivered = Boolean
Boolean = Message.Redelivered
```

The ReplyTo Property

The ReplyTo property sets or returns were a reply to this message will be sent. Destination could be a Topic, Queue, TemporaryTopic, or a TemporaryQueue.

```
Message.ReplyTo = Destination
Destination = Message.ReplyTo
```

The Timestamp Property

The TimeStamp property sets or returns the message timestamp.

```
Message.Timestamp = Currency
Currency = Message.Timestamp
```

The Type Property

The Type property sets or returns the message type.

```
Message.Type = String
String = Message.Type
```

5.1.13. The MessageConsumer Object

The MessageConsumer receives messages from a destination. This is an abstract interface.

The Close Method

The Close method closes resources on behalf of a MessageConsumer. A Message Service may allocate resources on behalf of a MessageConsumer, it is recommended that you close any unused resources.

```
MessageConsumer.Close
```

The Receive Message Method

The ReceiveMessage method receives the next message produced or that arrives within the specified timeout interval for this message consumer.

```
MessageConsumer.Receive([timeOut]) As message
```

Name	Description
timeout	The timeout value (in milliseconds) of the MessageConsumer.

The ReceiveNoWait Method

The ReceiveNoWait method receives the next message if one is immediately available.

```
MessageConsumer.ReceiveNoWait() As message
```

5.1.14. Properties of the MessageConsumer Object

The MessageListener Property

This property is currently not supported.

The MessageSelector Property

The MessageSelector property returns this message consumer's message selector expression.

```
MessageConsumer.MessageSelector = String
String = MessageConsumer.MessageSelector
```

5.1.15. The MessageListener Object

This object is currently not supported.

The OnMessage Property

This function is currently not supported.

5.1.16. The MessageProducer Object

The MessageProducer sends messages to a destination. Sub interfaces of the MessageProducer Object include QueueSender and TopicPublisher. This is an abstract interface.

There are no methods associated with this object.

5.1.17.Properties of the MessageProducer Object

The DeliveryMode Property

The DeliveryMode property sets or returns the delivery mode for this message as either msNonPersistent, or msPersistent. The default value is msDefaultDeliveryMode (msPersistent).

```
DeliveryMode = MessageProducerConstant
MessageProducerConstant = DeliveryMode
```

Name	Description
msPersistent	This mode instructs the JMS IQ Manager to log the message to storage as part of the client's send operation.
msNon_Persistent	This is the lowest overhead delivery mode because it does not require the message to be logged to storage. the JMS IQ Manager delivers a Non_Persistent message with a one-time-only delivery. If the JMS IQ Manager goes down, the message will be lost.

The DisableMessageID Property

The DisableMessageID property sets or returns whether message IDs are disabled.

```
MessageProducer.DisableMessageID = Boolean
Boolean = MessageProducer.DisableMessageID
```

The DisableMessageTimestamp Property

The DisableMessageTimestamp property sets or returns whether a messages timestamps are disabled.

```
MessageProducer.DisableMessageTimestamp = Boolean
Boolean = MessageProducer.DisableMessageTimestamp
```

The Priority Method

Currently not supported.

The TimeToLive Method

Returns or sets the default length of time in milliseconds from its dispatch time that a produced message should be retained by the message system, default value is msDefaultTimeToLive i.e. zero which is unlimited.

```
MessageProducer.TimeToLive = Currency
Currency = MessageProducer.TimeToLive
```

5.1.18. The Queue Object

A Queue object encapsulates a Message Service specific queue name.

The ToString Method

The ToString method returns a printed version of the queue name.

```
Queue.ToString() As String
```

5.1.19. Properties of the Queue Object

The QueueName Property

Returns the name of this queue. Read-only.

5.1.20. The QueueBrowser Object

This object is currently not supported.

5.1.21. The QueueConnection Object

A QueueConnection is an active connection to a PTP Message Service.

The CreateQueueSession Method

Create a QueueSession, where the possible values of acknowledgeMode are: msAutoAcknowledge, msClientAcknowledge and msDupsOkAcknowledge.

```
QueueConnection.CreateQueueSession(Transacted As Boolean,
acknowledgeMode As AcknowledgeModeConstants) As QueueSession
```

Name	Description
Transacted	If true, session is transacted.

Name	Description
acknowledgeMode	**msAutoAcknowledge** = 1 : The session automatically acknowledges a client's receipt of a message when it has either successfully returned from a call to **receive** or the MessageListener it has called to process the message successfully returns. **msClientAcknowledge** = 2 : A client acknowledges a message by calling the message's **acknowledge** method. Acknowledging a consumed message automatically acknowledges the receipt of all messages that have been delivered by its session. **msDupsOkAcknowledge** = 3 : Instructs the session to lazily acknowledge the delivery of messages. This is likely to result in the delivery of some duplicate messages if the Message Service fails. It should only be used by consumers that are tolerant of duplicate messages. The benefit is the reduction of session overhead, achieved by minimizing the work done to prevent duplicates.

The Start Method

Start (or restart) a Connection's delivery of incoming messages.

```
QueueConnection.Start
```

The Stop Method

Used to temporarily stop a Connection's delivery of incoming messages.

```
QueueConnection.Stop
```

5.1.22. Properties of QueueConnection Object

The ClientID Property

Returns or sets client identifier for this connection.

```
QueueConnection.ClientID = String
String = QueueConnection.ClientID
```

The MetaData Property

Not currently supported.

5.1.23. The QueueConnectionFactory Object

A client uses a QueueConnectionFactory to create QueueConnections with a PTP Message Service.

The CreateQueueConnection Method

Create a queue connection with a default user identity.

```
QueueConnectionFactory.CreateQueueConnection() As QueueConnection
```

5.1.24. Properties of the QueueConnectionFactory Object

The HostName Property

Returns or sets host name of the machine where Message Service is running.

```
QueueConnectionFactory.HostName = String
String = QueueConnectionFactory.HostName
```

The Port Property

Returns or sets port number at which Message Service is listening, default value is 24053.

```
QueueConnectionFactory.Port = Long
Long = QueueConnectionFactory
```

The PortOffset Property

Returns or sets port offset number of Message Service if more then one Message Service is running on same host machine and using same port number.

```
QueueConnectionFactory.PortOffset = Long
Long = QueueConnectionFactory.PortOffset
```

5.1.25. The QueueReceiver Object

A client uses a QueueReceiver for receiving messages that have been delivered to a queue.

The Close Method

Since a Message Service may allocate some resources on behalf of a MessageConsumer, you should close them when they are not needed.

```
QueueReceiver.Close
```

The Receive Method

Receive the next message produced or that arrives within the specified timeout interval for this message consumer

```
QueueReceiver.Receive([timeOut]) As message
```

Name	Description
timeout	The timeout value (in milliseconds) of the MessageConsumer.

The ReceiveNoWait Method

Receive the next message if one is immediately available.

```
QueueReceiver.ReceiveNoWait As message
```

5.1.26. Properties of the QueueReceiver Object

The MessageListener Property

This property is not currently supported.

The MessageSelector Property

Returns this message consumer's message selector expression.

```
QueueReceiver.MessageSelector = String
String = QueueReceiver.MessageSelector
```

The Queue Property

Returns the queue associated with this queue receiver.

```
QueueReceiver.Queue = Queue read only
Queue read only = QueueReceiver.Queue
```

5.1.27. The QueueRequestor Object

The QueueRequestor object provides a helper class to simplify making service requests.

The Create Method

Constructs the QueueRequestor.

```
QueueRequestor.Create(session As QueueSession, Queue As Queue)
```

Name	Description
session	The QueueSession.
queue	Queue name.

The Request Method

The Request method sends a request and waits for a reply.

```
QueueRequestor.Request(message As message) As message
```

Name	Description
message	The message.

5.1.28. The QueueSender Object

A client uses a QueueSender to send messages to a queue.

The Send Method

Sends a message to a queue for an unidentified message producer, specifying delivery mode, priority and time to live.

```
QueueSender.Send(message As message, [DeliveryMode], [Priority],
[TimeToLive], [Queue])
```

Name	Description
message	The message to be sent.
deliveryMode	The delivery mode to use.
priority	The priority for this message. Although not currently supported, it is suggested that you include the priority so as not to have to modify the code at a later date.
timeToLive	The message's lifetime (in milliseconds).
queue	The queue that this message should be sent to.

5.1.29. Properties of the QueueSender Object

The DeliveryMode Property

The DeliveryMode property sets or returns the delivery mode for this message as either msNonPersistent, or msPersistent. The default value is msDefaultDeliveryMode (msPersistent).

```
DeliveryMode= DeliveryModeConstant
DeliveryModeConstant = DeliveryMode
```

Name	Description
msPersistent	This mode instructs the JMS IQ Manager to log the message to storage as part of the client's send operation.

Name	Description
msNon_Persistent	This is the lowest overhead delivery mode because it does not require the message to be logged to storage. the JMS IQ Manager delivers a Non_Persistent message with a one-time-only delivery. If the JMS IQ Manager goes down, the message will be lost.

The DisableMessageID Property

Returns or sets an indication of whether message IDs are disabled

```
QueueSender.DisableMessageID = Boolean
Boolean = QueueSender.DisableMessageID
```

The DisableMessageTimestamp Property

Returns or sets an indication of whether message timestamps are disabled.

```
QueueSender.DisableMessageTimestamp = Boolean
Boolean = QueueSender.DisableMessageTimestamp
```

The Priority Property

Currently not supported. It is recommended that you pass in the parameter as if supported, to prevent the need to modify code at a later date.

The Queue Property

Returns the queue associated with this queue sender (read-only).

```
QueueSender.Queue = read only
read only = QueueSender.Queue
```

The TimeToLive Property

Returns or sets the default length of time in milliseconds, from its dispatch time that a produced message should be retained by the message system. The default value is msDefaultTimeToLive, zero, which is unlimited.

```
QueueSender.TimeToLive = Currency
Currency = QueueSender.TimeToLive
```

5.1.30. The QueueSession Object

A QueueSession provides methods for creating QueueReceivers, QueueSenders, QueueBrowsers, and TemporaryQueues.

The Commit Method

Commit all messages done in this transaction and releases any locks currently held.

```
QueueSession.Commit
```

The CreateBrowser Method

Create a QueueBrowser to peek at the messages on the specified queue

```
QueueSession.CreateBrowser.(Queue As Queue, [MessageSelector]) As
QueueBrowser
```

Name	Description
queue	The queue to access.
messageSelector	Only messages with properties matching the message selector expression are delivered.

The CreateBytesMessage Method

Create a BytesMessage.

```
QueueSession.CreateBytesMessage() As BytesMessage
```

The CreateMapMessage Method

Create a MapMessage.

```
QueueSession.CreateMapMessage() As MapMessage
```

The CreateMessage Method

Create a Message.

```
QueueSession.CreateMessage() As message
```

The CreateQueue Method

Create a queue identity given a Queue name.

```
QueueSession.CreateQueue(QueueName As String) As Queue
```

Name	Description
QueueName	The name of the queue.

The CreateReceiver Method

Create a QueueReceiver to receive messages for the specified queue.

```
QueueSession.CreateReceiver(Queue As Queue, [MessageSelector]) As
QueueReceiver
```

Name	Description
Queue	The queue to access.
MessageSelector	Only messages with properties matching the message selector expression are delivered.

The CreateSender Method

Create a QueueSender to send messages to the specified queue.

```
QueueSession.CreateSender(Queue As Queue) As QueueSender
```

Name	Description
Queue	The name of the queue.

The CreateStreamMessage Method

Create a StreamMessage.

```
QueueSession.StreamMessage() As StreamMessage
```

The CreateTemporaryQueue Method

Create a temporary queue.

```
QueueSession.CreateTemporaryQueue() As TemporaryQueue
```

The CreateTextMessage Method

Create a TextMessage.

```
QueueSession.CreateTextMessage([Text]) As TextMessage
```

Name	Description
Text	The string used to initialize this message.

The Recover Method

Stops message delivery int his session, and restart sending messages with the oldest unacknowledged message.

```
QueueSession.Recover()
```

The Rollback Method

Rolls back any messages done in this transaction and releases any lock currently held.

```
QueueSession.Rollback()
```

The Run Method

Only intended to be used by Application Servers (optional operation).

```
QueueSession.Run()
```

5.1.31. Properties of the QueueSender Object

The MessageListener Property

This property is not currently supported.

The Transacted Property

Returns an indication that the session is in transacted mode.

```
QueueSession.Transacted = Boolean
Boolean = QueueSession.Transacted
```

5.1.32. The Session Object

The Session object is a single threaded context for producing and consuming messages

The Commit Method

Commit all messages done in this transaction and releases any locks currently held.

```
Session.Commit
```

The CreateBytesMessage Method

The CreateBytesMessage method creates a BytesMessage.

```
Session.CreateBytesMessage() As BytesMessage
```

The CreateMapMessage Method

The CreateMapMessage method creates a MapMessage.

```
Session.CreateMapMessage() As MapMessage
```

The CreateMessage Method

Create a Message.

```
Session.CreateMessage() As message
```

The CreateStreamMessage Method

Create a StreamMessage.

```
Session.CreateStreamMessage() As StreamMessage
```

The CreateTextMessage Method

Create a TextMessage.

```
Session.CreateTextMessage([Text])
```

Name	Description
Text	The string used to initialize this message.

The Recover Method

The Recover method stops message delivery in this session, and restarts sending messages beginning with the oldest unacknowledged message.

```
Session.Recover
```

The Rollback Method

The Rollback method rollbacks any messages done in this transaction and releases any locks currently held.

```
Session.Rollback
```

The Run Method

The Run method is an optional operation that is only intended to be used by the JMS IQ Manager.

```
Session.Run
```

5.1.33. Properties of the Session Object

The MessageListener Property

This property is currently not supported.

The Transacted Property

The Transacted property returns an indication that the session is in transacted mode.

```
Session.Transacted = Boolean
Boolean = Session.Transacted
```

5.1.34. The StreamMessage Object

The StreamMessage object is used to send a stream of primitive data types.

The Acknowledge Method

Acknowledges the receipt of current and previous messages.

```
StreamMessage.acknowledge
```

The ClearBody Method

Clears the body of a message, leaving the message header values and property entries intact.

```
StreamMessage.ClearBody
```

The ClearProperties Method

Clears the properties from a message, leaving the message header fields and body intact.

```
StreamMessage.ClearProperties
```

The GetProperty Method

Returns the Visual Basic data type property value with the given name, into the Message.

```
StreamMessage.GetProperty(name As String)
```

Name	Description
name	The name of the property.

The PropertyExists Method

Checks whether a value for a specific property exists.

```
StreamMessage.PropertyExists(name As String) As Boolean
```

Name	Description
name	The name of the property to check.

The ReadBoolean Method

Reads a Boolean value from the bytes message stream.

```
StreamMessage.ReadBoolean() As Boolean
```

The ReadByte Method

Reads a signed 8-bit value from the bytes message stream.

```
StreamMessage.ReadByte() As Byte
```

The ReadBytes Method

Reads a portion of the bytes message stream.

```
StreamMessage.ReadBytes(value, [length As Long]) As Long
```

Name	Description
value	The buffer the data is read into.
length	The number of bytes array read. This number must be less than or equal to value.length.

The ReadChar Method

Reads a Unicode character value from the bytes message stream.

```
StreamMessage.ReadChar() As Integer
```

The ReadDouble Method

Reads a double from the bytes message stream.

```
StreamMessage.ReadDouble() As Double
```

The ReadFloat Method

Reads a float from the bytes message stream.

```
StreamMessage.ReadFloat() As Single
```

The ReadInt Method

Reads a signed 32-bit integer from the bytes message stream.

```
StreamMessage.ReadInt() As Long
```

The ReadLong Method

Reads a signed 64-bit integer from the bytes message stream.

```
SteamMessage.ReadLong() As Currency
```

The ReadObject Method

Currently not supported.

The ReadShort Method

Reads a signed 16-bit number from the bytes message stream.

```
StreamMessage.ReadShort() As Integer
```

The ReadString Method

The ReadString method reads in a string from the stream message.

```
StreamMessage.ReadString() As String
```

The Reset Method

The Reset method puts the message body in read-only mode, and repositions the stream of bytes to the beginning.

```
StreamMessage.Reset
```

The SetProperty Method

Set a Visual Basic data type property value with the given name, into the Message.

```
StreamMessage.SetProperty(name As String, value)
```

Name	Description
name	The name of the property.
value	The value to set.

The WriteBoolean Method

WriteBoolean writes to the bytes message stream as a 1-byte value.

```
StreamMessage.WriteBoolean(value as Boolean)
```

Name	Description
value	The boolean value to be written.

The WriteByte Method

WriteByte writes to the bytes message stream as a 1-byte value

```
StreamMessage.WriteByte(value As Byte)
```

Name	Description
value	The byte value to be written.

The WriteBytes Method

WriteBytes writes a byte array or string to the bytes message stream

```
StreamMessage.WriteBytes(value, [offset], [length])
```

Name	Description
value	The byte array value to be written.
offset	The initial offset within the byte array.
length	The number of bytes to use.

The WriteChar Method

WriteChar writes a char to the bytes message stream as a 2-byte value, high byte first

```
StreamMessage.WriteChar(value As Integer)
```

Name	Description
value	The char value to be written.

The WriteDouble Method

Uses the doubleToLongBits method (class Double) to convert the double parameter value to a long, and then writes an 8-byte long value to the bytes message stream (high byte is written first).

```
StreamMessage.WriteDouble(value As Double)
```

Name	Description
value	The double value to write to the message stream.

The WriteFloat Method

Convert the float argument to an long, and then writes that long value to the bytes message stream as a 4-byte quantity, high byte first

```
StreamMessage.WriteFloat(value As Single)
```

Name	Description
value	The float value to be written.

The WriteInt Method

Write an int to the bytes message stream as four bytes, high byte first.

```
StreamMessage.WriteInt(value As Long)
```

Name	Description
value	The int value to be written.

The WriteLong Method

WriteLong writes a long to the bytes message stream as eight bytes, high byte first

```
StreamMessage.WriteLong(value As Currency)
```

Name	Description
value	The long value to be written as currency.

The WriteObject Method

Currently not supported

The WriteShort Method

WriteShort writes a short to the bytes message stream as two bytes, high byte first

```
StreamMessage.WriteShort(value As Integer)
```

Name	Description
value	The short that is written.

The WriteString Method

Write a string to the message stream.

```
StreamMessage.WriteString(value as String)
```

Name	Description
value	The String value that is written.

5.1.35. Properties of the StreamMessage Object

The CorrelationID Property

The CorrelationID property sets or returns correlation id values that are either JMS IQ Manager specific message ID's or application-specific strings.

```
StreamMessage.CorrelationID = String
String = StreamMessage.CorrelationID
```

The CorrelationIDAsBytes Property

The CorrelationIDAsBytes property sets or returns the correlation ID as an array of bytes for the message.

```
StreamMessage.CorrelationIDAsBytes = Variant
Variant = StreamMessage.CorrelationIDAsBytes
```

The DeliveryMode Property

The DeliveryMode property sets or returns the delivery mode for this message as either msNonPersistent, or msPersistent. The default value is msDefaultDeliveryMode (msPersistent).

```
DeliveryMode = StreamMessageConstant
StreamMessageConstant = DeliveryMode
```

Name	Description
msDefaultDeliveryMode	Default DeliveryMode delivery mode.
msPersistent	This mode instructs the JMS IQ Manager to log the message to storage as part of the client's send operation.
msNon_Persistent	This is the lowest overhead delivery mode because it does not require the message to be logged to storage. the JMS IQ Manager delivers a Non_Persistent message with a one-time-only delivery. If the JMS IQ Manager goes down, the message will be lost.

The Destination Property

The Destination property sets or returns the destination for this message.

```
StreamMessage.Destination = Destination
Destination = StreamMessage.Destination
```

The Expiration Property

The Expiration property sets or returns the message expiration time in milliseconds.

```
StreamMessage.Expiration = Currency
Currency = StreamMessage.Expiration
```

The MessageID Property

The MessageID property sets or returns the value of the uniquely assigned identifier in the message header.

```
StreamMessage.MessageID = String
String = StreamMessage.MessageID
```

The Priority Property

The Priority property sets or returns the priority that is assigned to this message. Possible numbers are 1 - 9.

```
StreamMessage.Priority = PriorityConstant
PriorityConstant = StreamMessage.Priority
```

Name	Description
msPriorityZero through msPriorityNine	Level of a messages priority. Values are 0-9 with 0 being the lowest priority and 9 the highest. Four is the message priority default value.

The Redelivered Property

The Redelivered property sets or returns an indication as to whether this message is redelivered.

```
StreamMessage.Redelivered = Boolean
Boolean = StreamMessage.Redelivered
```

The ReplyTo Property

The ReplyTo property sets or returns were a reply to this message will be sent.

```
StreamMessage.ReplyTo = Destination
Destination = StreamMessage.ReplyTo
```

The Timestamp Property

The TimeStamp property sets or returns the message timestamp.

```
StreamMessage.Timestamp = Currency
Currency = StreamMessage.Timestamp
```

The Type Property

The Type property sets or returns the message type.

```
StreamMessage.Type = String
String = StreamMessage.Type
```

5.1.36. The TemporaryQueue Object

A TemporaryQueue is a unique Queue object created for the duration of a QueueConnection.

The Delete Method

The Delete method deletes the temporary queue.

```
TemporaryQueue.Delete
```

The ToString Method

The ToString method returns a printed version of the queue name

```
TemporaryQueue.ToString() As String
```

5.1.37. Properties of the TemporaryQueue Object

The QueueName Property

The QueueName property returns the name of this queue.

```
TemporaryQueue.QueueName = String
String = TemporaryQueue.QueueName
```

5.1.38. The TemporaryTopic Object

A TemporaryTopic is a unique Topic object created for the duration of a TopicConnection.

The Delete Method

The Delete method deletes the temporary topic.

```
TemporaryTopic.Delete
```

The ToString Method

The ToString method returns a printed version of the topic name

```
TemporaryTopic.ToString
```

5.1.39. Properties of the TemporaryTopic Object

The TopicName Property

The TopicName property returns the name of this topic.

```
TemporaryTopic.TopicName = String
String = TemporaryTopic.TopicName
```

5.1.40. The TextMessage Object

A TextMessage is used to send a message containing a String.

The Acknowledge Method

Acknowledges the receipt of current and previous messages.

```
TextMessage.acknowledge
```

The ClearBody Method

Clears the body of a message, leaving the message header values and property entries intact.

```
TextMessage.ClearBody
```

The ClearProperties Method

Clears the properties from a message, leaving the message header fields and body intact.

```
TextMessage.ClearProperties
```

The GetProperty Method

Returns the Visual Basic data type property value with the given name, into the Message.

```
TextMessage.GetProperty(name As String)
```

Name	Description
name	The name of the property.

The PropertyExists Method

Checks whether a value for a specific property exists.

```
TextMessage.PropertyExists(name As String) As Boolean
```

Name	Description
name	The name of the property to check.

The SetProperty Method

Set a Visual Basic data type property value with the given name, into the Message.

```
TextMessage.SetProperty(name As String, value)
```

Name	Description
name	The name of the byte property.
value	The value to set.

5.1.41. Properties of the Message Object

The CorrelationID Property

The CorrelationID property sets or returns correlation id values that are either JMS IQ Manager specific message ID's or application-specific strings.

```
TextMessage.CorrelationID = String
String = TextMessage.CorrelationID
```

The CorrelationIDAsBytes Property

The CorrelationIDAsBytes property sets or returns the correlation ID as an array of bytes for the message.

```
Message.CorrelationIDAsBytes = Variant
Variant = Message.CorrelationIDAsBytes
```

The DeliveryMode Property

The DeliveryMode property sets or returns the delivery mode for this message as either msNonPersistent, or msPersistent. The default value is msDefaultDeliveryMode (msPersistent).

```
DeliveryMode = BytesMessageConstant
BytesMessageConstant = DeliveryMode
```

Name	Description
msDefaultBytesMessage	Default BytesMessage delivery mode.
msPersistent	This mode instructs the JMS IQ Manager to log the message to storage as part of the client's send operation.
msNon_Persistent	This is the lowest overhead delivery mode because it does not require the message to be logged to storage. the JMS IQ Manager delivers a Non_Persistent message with a one-time-only delivery. If the JMS IQ Manager goes down, the message will be lost.

The Destination Property

The Destination property sets or returns the destination for this message.

```
TextMessage.Destination = Destination
Destination = TextMessage.Destination
```

The Expiration Property

The Expiration property sets or returns the message expiration time in milliseconds.

```
Message.Expiration = Currency
Currency = Message.Expiration
```

The MessageID Property

The MessageID property sets or returns the value of the uniquely assigned identifier in the message header.

```
TextMessage.MessageID = String
String = TextMessage.MessageID
```

The Priority Property

The Priority property sets or returns the priority that is assigned to this message. Possible numbers are 1 - 9. (Not currently supported, but suggested that the value be entered, to prevent code changes later.)

```
TextMessage.Priority = PriorityConstant
PriorityConstant = TextMessage.Priority
```

Name	Description
msPriorityZero through msPriorityNine	Level of a messages priority. Values are 0-9 with 0 being the lowest priority and 9 the highest. Four is the message priority default value.

The Redelivered Property

The Redelivered property sets or returns an indication as to whether this message is redelivered.

```
TextMessage.Redelivered = Boolean
Boolean = TextMessage.Redelivered
```

The ReplyTo Property

The ReplyTo property sets or returns where a reply to this message will be sent.

```
TextMessage.ReplyTo = Destination
Destination = TextMessage.ReplyTo
```

The Text Property

The Text property sets or returns the string containing the message's data.

```
TextMessage.Text = String
String = TextMessage.Text
```

The Timestamp Property

The TimeStamp property sets or returns the message timestamp.

```
TextMessage.Timestamp = Currency
Currency = TextMessage.Timestamp
```

The Type Property

The Type property sets or returns the message type.

```
TextMessage.Type = String
String = TextMessage.Type
```

5.1.42. The Topic Object

A Topic object encapsulates a Message Service specific topic name.

The ToString Method

The ToString method returns a printed version of the topic name

```
Topic.ToString() As String
```

5.1.43. Properties of the Topic Object

The TopicName Property

The TopicName property returns the name of this topic.

```
Topic.TopicName = String
String = Topic.TopicName
```

5.1.44. The TopicConnection Object

A TopicConnection is an active connection to a Pub/Sub Message Service.

The CreateTopicSession Method

Create a TopicSession

```
TopicConnection.CreateTopicSession(Transacted As Boolean,
acknowledgeMode As AcknowledgeModeConstants) As TopicSession
```

Name	Description
Transacted	If true, session is transacted.
acknowledgeMode	**msAutoAcknowledge** = 1 : The session automatically acknowledges a client's receipt of a message when it has either successfully returned from a call to receive or the MessageListener it has called to process the message successfully returns. **msClientAcknowledge** = 2 : A client acknowledges a message by calling the message's acknowledge method. Acknowledging a consumed message automatically acknowledges the receipt of all messages that have been delivered by its session. **msDupsOkAcknowledge** = 3 : Instructs the session to lazily acknowledge the delivery of messages. This is likely to result in the delivery of some duplicate messages if the Message Service fails. It should only be used by consumers that are tolerant of duplicate messages. The benefit is the reduction of session overhead, achieved by minimizing the work done to prevent duplicates.

The Start Method

The Start method starts or restarts a connection's delivery of incoming messages.

```
TopicConnection.Start
```

The Stop Method

The Stop method temporarily stops a Connection's delivery of incoming messages.

TopicConnection.Stop

5.1.45. Properties of the TopicConnection

The ClientID Property

The ClientID property sets or returns a client identifier for this connection.

```
TopicConnection.ClientID = String
String = TopicConnection.ClientID
```

The MetaData Property

This property is currently not supported.

5.1.46. The TopicConnectionFactory Object

A client uses a TopicconnectionFactory to create TopicConnection with a Pub/Sub Message Service.

The CreateTopicConnection Method

Create a topic connection with default user identity.

```
TopicConnectionFactory.CreateTopicConnection() As TopicConnection
```

5.1.47. Properties of the TopicConnectionFactory

The HostName Property

The HostName property sets or returns the host name of the machine that the JMS IQ Server is running on.

```
TopicConnectionFactory.HostName = String
String = TopicConnectionFactory.HostName
```

The Port Property

The Port property sets or returns the port number that the JMS IQ Server is listening on, default value is 18008

```
TopicConnectionFactory = Long
Long = TopicConnectionFactory
```

The PortOffset Property

The PortOffset sets or returns the port offset number of the JMS IQ Server if more then one Message Service is running on same host machine and using same port number.

```
TopicConnectionFactory.PortOffset = Long
Long = TopicConnectionFactory
```

5.1.48. The TopicPublisher Object

A Client uses a TopicPublisher for publishing messages on a topic.

The Publish Method

The Publish method publishes a Message to a topic for an unidentified message producer, specifying delivery mode, priority and time to live.

```
TopicPublisher.Publish(message As message, [DeliveryMode],
[Priority], [TimeToLive], [Topic])
```

Name	Description
message	The message to publish.
deliveryMode	The delivery mode to use.
priority	The priority for this message. While not currently supported, it is recommended to implement now, to prevent code changes later.
timeToLive	The message's lifetime (in milliseconds).
topic	The topic to publish this message to.

5.1.49. Properties of TopicPublisher

The DeliveryMode Property

The DeliveryMode property sets or returns the delivery mode for this message as either msNonPersistent, or msPersistent. The default value is msDefaultDeliveryMode (msPersistent).

```
DeliveryMode = BytesMessageConstant
BytesMessageConstant = DeliveryMode
```

Name	Description
msPersistent	This mode instructs the JMS IQ Manager to log the message to storage as part of the client's send operation.
msNon_Persistent	This is the lowest overhead delivery mode because it does not require the message to be logged to storage. the JMS IQ Manager delivers a Non_Persistent message with a one-time-only delivery. If the JMS IQ Manager goes down, the message will be lost.

The DisableMessageID Property

The DisableMessageID property sets or returns whether message IDs are disabled.

```
TopicPublisher.DisableMessageID = Boolean
Boolean = TopicPublisher.DisableMessageID
```

The DisableMessageTimestamp Property

The DisableMessageTimestamp sets or returns an indication of whether message timestamps are disabled.

```
TopicPublisher.DisableMessageTimestamp = Boolean
Boolean = TopicPublisher.DisableMessageTimestamp
```

The Priority Property

The Priority property sets or returns the priority that is assigned to this message. Possible numbers are 1 - 9. While not currently supported, it is suggested that the desired value be entered now, to prevent code changes later.

```
TopicPublisher.Priority = PriorityConstant
PriorityConstant = TopicPublisher.Priority
```

Name	Description
msPriorityZero through msPriorityNine	Level of a messages priority. Values are 0-9 with 0 being the lowest priority and 9 the highest. Four is the message priority default value.

The TimeToLive Property

The TimeToLive property sets and returns the default length of time in milliseconds from its dispatch time that a produced message should be retained by the message system.

```
TopicPublisher.TimeToLive = MessageConstant
MessageConstant = TopicPublisher.TimeToLive
```

Name	Description
msDefaultTimeToLive	The default value of 0 = Unlimited

The Topic Property

The Topic property returns the topic associated with this publisher.

```
TopicPublisher.Topic = read-only
read-only = TopicPublisher.Topic
```

5.1.50. The TopicRequestor Object

The TopicRequestor object provides a helper class to simplify making service requests.

The Create Method

Constructs the TopicRequestor.

```
TopicRequestor.Create(session As TopicSession, Topic As Topic)
```

Name	Description
session	The name of the topic session.
topic	The name of the topic.

The Request Method

Send a request and wait for a reply

```
TopicRequestor.Request(message As message) As message
```

Name	Description
message	The message text.

5.1.51. The TopicSession Object

A TopicSession provides methods for creating TopicPublishers, TopicSubscribers, and TemporaryTopics.

The Commit Method

The Commit method commits all messages done in this transaction and releases any resources, currently held.

```
TopicSession.Commit
```

The CreateBytesMessage Method

The CreateBytesMessage method creates a BytesMessage.

```
TopicSession.CreateBytesMessage() As BytesMessage
```

The CreateDurableSubscriber Method

The CreateDurableSubscriber method creates a durable Subscriber to the specified topic.

```
TopicSession.CreateDurableSubscriber(Topic As Topic, name As String,
[MessageSelector], [NoLocal] As TopicSubscriber
```

Name	Description
topic	The non-temporary topic to subscribe to.
name	The name used to identify this subscription.
messageSelector	Only messages with properties matching the message selector expression are delivered. You may use a null.
noLocal	If set, noLocal inhibits the delivery of messages published by its own connection.

The CreateMapMessage Method

The CreateMapMessage method creates a MapMessage.

```
TopicSession.CreateMapMessage() As MapMessage
```

The CreateMessage Method

Create a Message.

```
TopicSession.CreateMessage() As message
```

The CreatePublisher Method

Create a Publisher for the specified topic.

```
TopicSession.CreatePublisher(Topic As Topic) As TopicPublisher
```

Name	Description
topic	The topic to which to publish, or null, if this is an unidentified producer.

The CreateStreamMessage Method

Create a StreamMessage.

```
TopicSession.CreateStreamMessage() As StreamMessage
```

The CreateSubscriber Method

Create a non-durable Subscriber to the specified topic

```
TopicSession.CreateSubscriber(Topic As Topic, [MessageSelector],
[NoLocal]) As TopicSubscriber
```

Name	Description
topic	The topic to subscribe to.
messageSelector	Only messages with properties matching the message selector expression are delivered. This value may be null.
noLocal	If set, inhibits the delivery of messages published by its own connection.

The CreateTemporaryTopic Method

The CreateTemporaryTopic method creates a temporary topic.

```
TopicSession.CreateTemporaryTopic() As TemporaryTopic
```

The CreateTextMessage Method

The CreateTextMessage method creates TextMessage.

```
TopicSession.CreateTextMessage([Text]) As TextMessage
```

Name	Description
text	The string used to initialize this message.

The CreateTopic Method

Create a topic identity given a Topic name.

```
TopicSession.CreateTopic(TopicName As String) As Topic
```

Name	Description
topicName	The name of this topic.

The Recover Method

The Recover method creates a topic identity given a Topic name.

```
TopicSession.Recover
```

The Rollback Method

The Rollback method rolls back any messages done in this transaction and releases any locks currently held.

```
TopicSession.Rollback
```

The Run Method

The Run method is an optional method

```
TopicSession.Run
```

The Unsubscribe Method

The Unsubscribe method unsubscribes a durable subscription that has been created by a client.

```
TopicSession.Unsubscribe(name As String)
```

Name	Description
name	The name used to identify this subscription.

5.1.52. Properties of the TopicSession Object

The MessageListener Property

This property is currently not supported.

The Transacted Property

The Transacted property returns an indication that the session is in transacted mode.

```
TopicSession.Transacted = Boolean
Boolean = TopicSession.Transacted
```

5.1.53. The TopicSubscriber Object

A client uses a TopicSubscriber for receiving messages that have been published to a topic.

The Close Method

Since a Message Service may allocate resources on behalf of a MessageConsumer, clients should close any unneeded resources.

```
TopicSubscriber.Close
```

The Receive Method

The Receive method receives the next message produced or that arrives within the specified timeout interval for this message consumer

```
TopicSubscriber.Receive([timeOut]) As message
```

Name	Description
timeout	The timeout value (in milliseconds).

The ReceiveNoWait Method

The ReceiveNoWait method receives the next message if one is immediately available.

```
TopicSubscriber.ReceiveNoWait() As message
```

5.1.54. Properties of the TopicSubscriber Object

The MessageListener Property

This property is currently not supported.

The MessageSelector Property

The MessageSelector property returns this message consumer's message selector expression.

```
TopicSubscriber.MessageSelector = String
String = TopicSubscriber.MessageSelector
```

The NoLocal Property

The NoLocal property returns the NoLocal attribute for this TopicSubscriber.

```
TopicSubscriber.NoLocal = Boolean
Boolean = TopicSubscriber.NoLocal
```

The Topic Property

The Topic property returns the topic associated with this subscriber.

```
TopicSubscriber.Topic = Topic (read-only)
Topic (read-only) = TopicSubscriber.Topic
```

5.1.55. The XAQueueConnection Object

An XAQueueConnection provides the same create options as QueueConnection. The only difference is that an XAQueueConnection is by definition transacted.

The CreateQueueSession Method

Create a QueueSession, where the possible values of acknowledgeMode are: msAutoAcknowledge, msClientAcknowledge and msDupsOkAcknowledge.

```
XAQueueConnection.CreateQueueSession(Transacted As Boolean,
acknowledgeMode As AcknowledgeModeConstants) As QueueSession
```

Name	Description
Transacted	If true, session is transacted.
acknowledgeMode	**msAutoAcknowledge** = 1 : The session automatically acknowledges a client's receipt of a message when it has either successfully returned from a call to receive or the MessageListener it has called to process the message successfully returns. **msClientAcknowledge** = 2 : A client acknowledges a message by calling the message's acknowledge method. Acknowledging a consumed message automatically acknowledges the receipt of all messages that have been delivered by its session. **msDupsOkAcknowledge** = 3 : Instructs the session to lazily acknowledge the delivery of messages. This is likely to result in the delivery of some duplicate messages if the Message Service fails. It should only be used by consumers that are tolerant of duplicate messages. The benefit is the reduction of session overhead, achieved by minimizing the work done to prevent duplicates.

The CreateXAQueueSession Method

Create an XAQueueSession.

```
XAQueueConnection.CreateXAQueueSession() As XAQueueSession
```

The Start Method

Start (or restart) a Connection's delivery of incoming messages.

```
XAQueueConnection.Start
```

The Stop Method

Used to temporarily stop a Connection's delivery of incoming messages.

```
XAQueueConnection.Stop
```

5.1.56. Properties of XAQueueConnection Object

The ClientID Property

Returns or sets client identifier for this connection.

```
XAQueueConnection.ClientID = String
String = XAQueueConnection.ClientID
```

The MetaData Property

Not currently supported.

5.1.57. The XAQueueConnectionFactory Object

An XAQueueConnectionFactory provides the same create options as a QueueConnectionFactory, by definition, it is transacted.

The CreateQueueConnection Method

Create a queue connection with a default user identity.

```
XAQueueConnectionFactory.CreateQueueConnection() As QueueConnection
```

The CreateXAQueueConnection Method

Create an XA queue connection with a default user identity.

```
XAQueueConnectionFactory.CreateXAQueueConnection() As
XAQueueConnection
```

5.1.58. Properties of the QueueConnectionFactory Object

The HostName Property

Returns or sets host name of the machine where Message Service is running.

```
XAQueueConnectionFactory.HostName = String
String = XAQueueConnectionFactory.HostName
```

The Port Property

Returns or sets port number at which Message Service is listening, default value is 24053.

```
XAQueueConnectionFactory.Port = Long
Long = XAQueueConnectionFactory
```

The PortOffset Property

Returns or sets port offset number of Message Service if more then one Message Service is running on same host machine and using same port number.

```
XAQueueConnectionFactory.PortOffset = Long
Long = XAQueueConnectionFactory.PortOffset
```

5.1.59. The XAQueueSession Object

An XAQueueSession provides a regular QueueSession, which can be used to create QueueReceivers, QueueSenders, and QueueBrowsers.

The Commit Method

Commit all messages done in this transaction and releases any locks currently held.

```
XAQueueSession.Commit
```

The CreateBytesMessage Method

Create a BytesMessage.

```
XAQueueSession.CreateBytesMessage() As BytesMessage
```

The CreateMapMessage Method

Create a MapMessage.

```
XAQueueSession.CreateMapMessage() As MapMessage
```

The CreateMessage Method

Create a Message.

```
XAQueueSession.CreateMessage() As message
```

The CreateStreamMessage Method

Create a StreamMessage.

```
XAQueueSession.StreamMessage() As StreamMessage
```

The CreateTextMessage Method

Create a TextMessage.

```
XAQueueSession.CreateTextMessage([Text]) As TextMessage
```

Name	Description
Text	The string used to initialize this message.

The Recover Method

Stops message delivery int his session, and restart sending messages with the oldest unacknowledged message.

```
XAQueueSession.Recover()
```

The Rollback Method

Rolls back any messages done in this transaction and releases any lock currently held.

```
XAQueueSession.Rollback()
```

The Run Method

Only intended to be used by Application Servers (optional operation).

```
XAQueueSession.Run()
```

5.1.60. Properties of the QueueSender Object

The MessageListener Property

This property is not currently supported.

The QueueSession Property

Returns the queue session associated with this XAQueueSession.

```
XAQueueSession.QueueSession = QueueSession (read-only)
QueueSession (read-only)= XAQueueSession.QueueSession
```

The Transacted Property

Returns an indication that the session is in transacted mode.

```
XAQueueSession.Transacted = Boolean
Boolean = XAQueueSession.Transacted
```

5.1.61. The XASession Object

The XASession extends the capability of Session by adding access to a Message Service's support for Transaction, using the Compensating Resource Manager (CRM), handled under the Distributed Transaction Coordinator (DTC).

The Commit Method

Commit all messages done in this transaction and releases any locks currently held.

```
XASession.Commit
```

The CreateBytesMessage Method

The CreateBytesMessage method creates a BytesMessage.

```
XASession.CreateBytesMessage() As BytesMessage
```

The CreateMapMessage Method

The CreateMapMessage method creates a MapMessage.

```
XASession.CreateMapMessage() As MapMessage
```

The CreateMessage Method

Create a Message.

```
XASession.CreateMessage() As message
```

The CreateStreamMessage Method

Create a StreamMessage.

```
XASession.CreateStreamMessage() As StreamMessage
```

The CreateTextMessage Method

Create a TextMessage.

```
XASession.CreateTextMessage([Text])
```

Name	Description
Text	The string used to initialize this message.

The Recover Method

The Recover method stops message delivery in this session, and restarts sending messages beginning with the oldest unacknowledged message.

```
XASession.Recover
```

The Rollback Method

The Rollback method rollbacks any messages done in this transaction and releases any locks currently held.

```
XASession.Rollback
```

The Run Method

The Run method is an optional operation that is only intended to be used by the JMS IQ Manager.

```
XASession.Run
```

5.1.62. Properties of the Session Object

The MessageListener Property

This property is currently not supported.

The Transacted Property

The Transacted property returns an indication that the session is in transacted mode.

```
XASession.Transacted = Boolean
Boolean = XASession.Transacted
```

5.1.63. The XATopicConnection Object

An XATopicConnection provides the same create options as TopicConnection, but by definition is transacted.

The CreateTopicSession Method

Create a TopicSession

```
XATopicConnection.CreateTopicSession(Transacted As Boolean,
acknowledgeMode As AcknowledgeModeConstants) As TopicSession
```

Name	Description
Transacted	If true, session is transacted.
acknowledgeMode	**msAutoAcknowledge** = 1 : The session automatically acknowledges a client's receipt of a message when it has either successfully returned from a call to receive or the MessageListener it has called to process the message successfully returns. **msClientAcknowledge** = 2 : A client acknowledges a message by calling the message's acknowledge method. Acknowledging a consumed message automatically acknowledges the receipt of all messages that have been delivered by its session. **msDupsOkAcknowledge** = 3 : Instructs the session to lazily acknowledge the delivery of messages. This is likely to result in the delivery of some duplicate messages if the Message Service fails. It should only be used by consumers that are tolerant of duplicate messages. The benefit is the reduction of session overhead, achieved by minimizing the work done to prevent duplicates.

The Start Method

The Start method starts or restarts a connection's delivery of incoming messages.

```
XATopicConnection.Start
```

The Stop Method

The Stop method temporarily stops a Connection's delivery of incoming messages.

```
XATopicConnection.Stop
```

5.1.64. Properties of the TopicConnection

The ClientID Property

The ClientID property sets or returns a client identifier for this connection.

```
XATopicConnection.ClientID = String
String = XATopicConnection.ClientID
```

The MetaData Property

This property is currently not supported.

5.1.65. The XATopicConnectionFactory Object

An XATopicConnectionFactory provides the same create options as TopicConnectionFactory, but by definition is transacted.

The CreateTopicConnection Method

Create a topic connection with default user identity.

```
XATopicConnectionFactory.CreateTopicConnection() As TopicConnection
```

The CreateXATopicConnection Method

Create an XA topic connection with default user identity.

```
XATopicConnectionFactory.CreateTopicConnection() As TopicConnection
```

5.1.66. Properties of the TopicConnectionFactory

The HostName Property

The HostName property sets or returns the host name of the machine that the JMS IQ Server is running on.

```
XATopicConnectionFactory.HostName = String
String = XATopicConnectionFactory.HostName
```

The Port Property

The Port property sets or returns the port number that the JMS IQ Server is listening on, default value is 18008

```
XATopicConnectionFactory = Long
Long = XATopicConnectionFactory
```

The PortOffset Property

The PortOffset sets or returns the port offset number of the JMS IQ Server if more then one Message Service is running on same host machine and using same port number.

```
XATopicConnectionFactory.PortOffset = Long
Long = XATopicConnectionFactory
```

5.1.67. The XATopicSession Object

An XA TopicSession provides a regular TopicSession which can be used to create TopicSubscribers and TopicPublishers.

The Commit Method

The Commit method commits all messages done in this transaction and releases any resources, currently held.

```
XATopicSession.Commit
```

The CreateBytesMessage Method

The CreateBytesMessage method creates a BytesMessage.

```
XATopicSession.CreateBytesMessage() As BytesMessage
```

The CreateMapMessage Method

The CreateMapMessage method creates a MapMessage.

```
XATopicSession.CreateMapMessage() As MapMessage
```

The CreateMessage Method

Create a Message.

```
XATopicSession.CreateMessage() As message
```

The CreateStreamMessage Method

Create a StreamMessage.

```
XATopicSession.CreateStreamMessage() As StreamMessage
```

The CreateTextMessage Method

The CreateTextMessage method creates TextMessage.

```
XATopicSession.CreateTextMessage([Text]) As TextMessage
```

Name	Description
text	The string used to initialize this message.

The Recover Method

The Recover method creates a topic identity given a Topic name.

```
XATopicSession.Recover
```

The Rollback Method

The Rollback method rolls back any messages done in this transaction and releases any locks currently held.

```
TopicSession.Rollback
```

The Run Method

The Run method is an optional method

```
TopicSession.Run
```

5.1.68. Properties of the TopicSession Object

The MessageListener Property

This property is currently not supported.

The TopicSession Property

Returns the topic session associated with this XATopicSession.

```
XATopicSession.TopicSession = TopicSession (read-only)
TopicSession (read-only) = TopicSession.TopicSession
```

The Transacted Property

The Transacted property returns an indication that the session is in transacted mode.

```
TopicSession.Transacted = Boolean
Boolean = TopicSession.Transacted
```

5.2 COM+ Error Codes

Common error codes in COM+ APIs are given below.

IErrorInfo Methods

Table 6 IErrorInfo Methods

IErrorInfo	Methods Description
GetDescription	Returns a textual description of the error
GetGUID	Returns the globally unique identifier (GUID) for the interface that defined the error

HRESULT Errors

static HRESULT Error(LPCOLESTR lpszDesc, const IID& iid = GUID_NULL, HRESULT hRes = 0);

Description

This static method sets up the IErrorInfo interface to provide error information to the client. In order to call Error, your object must implement the ISupportErrorInfo interface.

If the hRes parameter is nonzero, then Error returns the value of hRes. If hRes is zero, then the first four versions of Error return DISP_E_EXCEPTION. The last two versions return the result of the macro MAKE_HRESULT(1, FACILITY_ITF, nID).

Table 7 Typical COM HRESULT Value

Value	Meaning
E_FAIL	Failure.
E_NOTIMPL	Method is not supported.
S_FALSE	Success. Condition was FALSE.
S_OK	Success. Numerically equivalent to NOERROR.

Error Codes

Error Code	Explanation
0x80040300 = 2147746560 decimal JE_CODE_E_GENERAL	JMS exception, unspecified.
0x80040301 = 2147746561 decimal JE_CODE_E_REALLOC	A JMS exception occurred as a result of memory reallocation.
0x80040302 = 2147746562 decimal JE_CODE_E_MALLOC	A JMS exception occurred as a result of memory allocation.
0x80040303 = 2147746563 decimal JE_CODE_E_CONNECTION	A JMS exception occurred in setting up a connection.
0x80040304 = 2147746564 decimal JE_CODE_E_CREATION	A JMS exception occurred while creating a JMS object.
0x80040305 = 2147746565 decimal JE_CODE_E_CLOSED_SOCKET	A JMS exception occurred because of a closed socket.
0x80040306 = 2147746566 decimal JE_CODE_E_EOF	Processing ended because the BytesMessage or StreamMessage ended unexpectedly.
0x80040307 = 2147746567 decimal JE_CODE_E_NOTREADABLE	Processing ended because the message could not be read.
0x80040308 = 2147746568 decimal JE_CODE_E_NOTWRITEABLE	Processing ended because the message could not be written.

Error Code	Explanation
0x80040309 = 2147746569 decimal JE_CODE_E_FORMAT	Processing ended because the JMS client attempted to use a data type not supported by the message (or a message property), or attempted to read message data (or a message property) as the wrong type.
0x8004030A = 2147746570 decimal JE_CODE_E_ROLLBACK	The attempt to commit the session was unsuccessful of a transaction being rolled back.
0x8004030B = 2147746571 decimal JE_CODE_E_STATE	Processing ended because a method was invoked at an illegal or inappropriate time or because the provider was not in an appropriate state for the requested operation.
0x8004030C = 2147746572 decimal JE_CODE_E_DESTINATION	Processing ended because the destination could not be understood or was found to be invalid.
0x8004030D = 2147746573 decimal JE_CODE_E_NOTIMPL	Processing ended because a feature or interface was not implemented.
0x8004030E = 2147746574 decimal JE_CODE_E_INDEX_OUT_OF_BOUNDS	Processing ended because an index of some sort (such as to an array, to a string, or to a vector) was found to be outside the valid range.
0x8004030F = 2147746575 decimal JE_CODE_E_NULL_POINTER	Processing ended because the pointer in a case where an object was required.
0x80040310 = 2147746576 decimal JE_CODE_E_INVLD_CLIENT_ID	Processing ended because the connection's client ID was rejected by the provider.
0x80040311 = 2147746577 decimal JE_CODE_E_INVLD_SELECTOR	Processing ended because the message selector was found to be syntactically invalid.
0x80040312 = 2147746578 decimal JE_CODE_E_JMS_SECURITY	Processing was ended by JMS Security — for example, the provider rejected a name/password combination submitted by a client.
0x80040313 = 2147746579 decimal JE_CODE_E_RESOURCE_ALLOC	Processing ended because of the provider was unable to allocate resources required for the method/function.
0x80040314 = 2147746580 decimal JE_CODE_E_XA_IN_PROGRESS	Processing ended because a transaction was in progress.

Chapter 6

Client Libraries for the SeeBeyond JMS API

This chapter provides a detailed discussion of the Application Program Interfaces (APIs) for the SeeBeyond Message Service.

For Java

- **The JMS API for the SeeBeyond Message Service** on page 142

The SeeBeyond Message Service includes additional API support in the following languages:

For COM+

- **The COM+ API for the SeeBeyond Message Service** on page 76

For C

- **The C API** on page 173
- **Error Codes and Messages in the C API** on page 273

For C++

The C++ API for the SeeBeyond Message Service on page 299

6.1 The JMS API for the SeeBeyond Message Service

6.1.1. The Standard Specification for the JMS API

The JMS API specification defined by Sun is available as a Javadoc. It can be downloaded from:

http://java.sun.com/products/jms/docs.html

Chapter 7

Implementing JMS in C

This chapter describes a sample implementation for the SeeBeyond Message Service using JMS in C.

What's in This Chapter

7.1 Implementing JMS Models in C

This section discusses how to use the JMS C APIs and their wrappers in C to exchange data with an eGate system.

7.1.1. Wrapper Functions for C

For C, the API Kit supplies a set of JMS wrapper functions. The purpose of this layer is to help developers focus on the programming task rather than on the details of JMS. Thus, while the wrapper functions are sufficient for most applications, they do not provide a complete function set; for details on the complete C API, see **"The C API" on page 173**.

At this higher level of abstraction, you need only manage a few types of structures:

- **Session**—Characterized by a hostname, port number, session type (either a pub/sub "topic session" or a point-to-point "queue session"), and connection parameters such as acknowledgment mode, transaction mode, delivery mode, and client ID.
- **Destination**—Either a topic (for pub/sub messaging) or else a queue (for point-to-point messaging). Characterized by session and destination name.
- Message **Producer**—Either a topic publisher or a queue sender. Characterized by session and destination.
- Message **Consumer**—Either a topic subscriber or a queue receiver. Characterized by session and destination (and, in pub/sub messaging only, by the name of the durable subscriber, if designated).

- **Requestor**—In request/reply messaging, either a topic requestor or a queue requestor. Characterized by session, destination, message, and time-to-live value expressed in milliseconds.
- **Message**—Characterized by the message type and payload. The payload is also called the message *body*, or message *content*:
  - For messages of type BytesMessage, the payload is an array of bytes.
  - For messages of type TextMessage, the payload is a string of text characters. Native encoding is used; for example, text on AS/400 systems uses EBCDIC encoding, but ASCII is used on most other systems.

For each of these structures, the wrapper provides the equivalent of a constructor (**Crt***Obj* or **Opn***Obj*) and a destructor (**Close***Obj* or **Dlt***Obj*). The other wrapper functions allow you to write/read/send/receive messages, to set up request/reply messaging, and to commit a transacted session.

The wrapper functions use parameters *err* and *errBuf* to handle error codes and error message text; see **"Differences Between Java and C in Error Handling" on page 274**.

7.1.2. Creating Destinations

Destinations do not need to be created separately: they are created through the `session.createQueue()` and `session.createTopic()` functions. If these destinations do not exist, they are created automatically.

7.2 Sample Code for Using JMS in C

From the **Samples** directory located on the eGate installation CD-ROM, navigate to the **jmsapi** folder. Select the **C** folder and extract the sample files from this folder to the computer that you have installed the eGate API Kit on. The samples were build using the compiler shown for the Windows operating system in **Table 1 on page 13**.

For the JMS API in C, the following sample programs are provided on the product CD-ROM:

- **Publish/Subscribe Messaging Using C** on page 145
- **Queue Messaging (Sending/Receiving) Using C** on page 150
- **Request-Reply Messaging Using C** on page 154
- **Message Selector Using C** on page 159
- **Publish/Subscribe Messaging Using XA in C** on page 164

Note: *For multi byte data processing on non-English operating systems, use the following method to set your locale leaving the double quote blank. This will allow the program to pick up your system's default locale setting.*

```
setlocale(LC_CTYPE,"");
```

7.2.1. Publish/Subscribe Messaging Using C

```
(1)    *------------------------------------------------------------------------
(2)    * Sample code to demonstrate JMS Pub/Sub Messaging.
(3)    *------------------------------------------------------------------------
(4)    *
(5)    *  Disclaimer:
(6)    *
(7)    *  Copyright 2002 by SeeBeyond Technology Corporation.
(8)    *  All Rights Reserved.
(9)    *   Unless otherwise stipulated in a written agreement for this
(10)   *  software, duly executed by SeeBeyond Technology Corporation, this
(11)   *  software is provided as is without warranty of any kind.  The
(12)   *  entire risk as to the results and performance of this software
(13)   *  is assumed by the user.SeeBeyond Technology Corporation disclaims
(14)   *  all warranties, either express or implied, including but not
(15)   *  limited, the implied warranties of merchantability, fitness for a
(16)   *  particular purpose, title and non-infringement, with respect to
(17)   *  this software.
(18)   *
(19)   * ------------------------------------------------------------------------*/
(20)  #include "mscapi.h"
(21)  #include <stdlib.h>
(22)  #include <stdio.h>
(23)  #include <string.h>
(24)  #ifndef WIN32
(25)  #include <unistd.h>
(26)  #endif
(27)  #if defined(WIN32)
(28)  #include "sbyn_getopt.h"
(29)  #endif
(30)
(31)  #if defined(OS400)
(32)  extern char *optarg;
(33)  #endif
(34)
(35)  #if defined(__gnu__)
(36)  #include <getopt.h>
(37)  #endif
(38)
(39)  char                    optionProducer[] = "[ -u ] [ -p port ]
(40)                          [ -h hostname ]";
(41)  char                    optionConsumer[] = "[ -c ] [ -p port ]
(42)                          [ -h hostname ]";
(43)  char                    optdescription[] = "\t-u   run as a
(44)                          producer\n\t-c run as a consumer\n\t-p
(45)                          port number\n\t-h   hostname\n";
(46)  static char             localHost[] = "localhost";
(47)  static unsigned short   susPort = 24053; /* default port number */
(48)  unsigned long           sulMessageSize = 16; /* default host name */
(49)  static char*            spHostName;
(50)  static void             subscriber();
(51)  static void             publisher();
(52)
(53)  /* Check for errors. */
(54)  static void check_error(int err, char* errBuf, int exitnow)
(55)  {
(56)       if (err){
(57)            printf("ERROR:0x%x - %s\n", err, errBuf);
(58)            if (exitnow)
(59)                 exit(1);
(60)       }
(61)  }
(62)
```

```
(63)  int main(int argc, char *argv[]) {
(64)       int           c;
(65)       char          cOption = 0;
(66)
(67)       spHostName = localHost;
(68)
(69)       while((c = getopt(argc, argv, ":p:h:P:H:cCUu")) != -1) {
(70)            switch(c){
(71)            case 'p':
(72)            case 'P':
(73)                 susPort = atoi(optarg);  /* setup the port number */
(74)                 break;
(75)            case 'h':
(76)            case 'H':
(77)                 spHostName = optarg;     /* setup the hostname */
(78)                 break;
(79)            case 'U':
(80)            case 'u':
(81)                 cOption = 'u';           /* run as a producer */
(82)                 break;
(83)            case 'c':
(84)            case 'C':
(85)                 cOption = 'c';           /* run as a consumer */
(86)                 break;
(87)            case ':':
(88)            case '?':
(89)                 printf("\nSYNOPSIS\n");
(90)                 printf("%s %s\n", argv[0], optionProducer);
(91)                 printf("%s %s\n", argv[0], optionConsumer);
(92)                 printf("%s\n", optdescription);
(93)                 exit(1);
(94)                 break;
(95)            }
(96)       }
(97)
(98)       if (cOption == 'u'){
(99)            publisher();          /* invoke producer */
(100)      } else if (cOption == 'c'){
(101)           subscriber();         /* invoke consumer */
(102)      } else {
(103)           printf("\nSYNOPSIS\n");
(104)           printf("%s %s\n", argv[0], optionProducer);
(105)           printf("%s %s\n", argv[0], optionConsumer);
(106)           printf("%s\n", optdescription);
(107)           exit(1);
(108)      }
(109) }
(110)
(111)
(112) /*
(113)  * ============================================================
(114)  * Topic Publisher
(115)  * This routine demonstrates how to publish to a topic.
(116)  * ============================================================
(117) */
(118) static void publisher() {
(119)      SBYN_TopicPublisher*    pTopicPublisher;
(120)      SBYN_Session*           pTopicSession;
(121)      SBYN_Destination*       pTopic;
(122)      SBYN_Message*           pTextMessage;
(123)      SBYN_Connection*        pTopicConnection;
(124)      SBYN_TopicConnectionFactory* pTcf;
(125)      int                     iErr;
(126)      char                    szErrBuf[256];
```

```
(127)      char                    pBuffer[] = "This is a text message";
(128)      char                    pTopicName[] = "PubSubSample";
(129)      int                     iMessagePriority = 4;
(130)      long                    iTimeToLive = 0;
(131)
(132)      /* Create a topic factory. */
(133)      pTcf = CreateTopicConnectionFactory(spHostName, susPort, &iErr,
(134)      szErrBuf);
(135)      check_error(iErr, szErrBuf, 1);
(136)      if(!pTcf) {
(137)           printf("CreateTopicConnectionFactory
(138)      failed\nError:%0X\nReason:%s\n", iErr, szErrBuf);
(139)           exit(2);
(140)      }
(141)
(142)      /* Create a topic connection. */
(143)      pTopicConnection = CreateTopicConnection(pTcf,&iErr, szErrBuf);
(144)      check_error(iErr, szErrBuf, 1);
(145)
(146)      /* Set the client ID. */
(147)      ConnectionSetClientID(pTopicConnection,
(148)      (char*)"TopicTestPublisher", &iErr, szErrBuf);
(149)      check_error(iErr, szErrBuf, 1);
(150)
(151)      /* Start the connection. */
(152)      ConnectionStart(pTopicConnection,&iErr, szErrBuf);
(153)      check_error(iErr, szErrBuf, 1);
(154)
(155)      /* Create a topic session. */
(156)      pTopicSession = ConnectionCreateTopicSession (pTopicConnection,
(157)      SBYN_TRANSACTED, SBYN_CLIENT_ACKNOWLEDGE, &iErr, szErrBuf);
(158)      check_error(iErr, szErrBuf, 1);
(159)      if(!pTopicSession) {
(160)           printf("CreateTopicSession failed\nError:%0X\nReason:%s\n",
(161)      iErr, szErrBuf);
(162)           exit(2);
(163)      }
(164)
(165)      /* Create a topic. */
(166)      pTopic = SessionCreateTopic(pTopicSession, pTopicName, &iErr,
(167)      szErrBuf);
(168)      check_error(iErr, szErrBuf, 1);
(169)
(170)      /* Create a topic publisher. */
(171)      pTopicPublisher = SessionCreatePublisher(pTopicSession, pTopic,
(172)      &iErr, szErrBuf);
(173)      check_error(iErr, szErrBuf, 1);
(174)
(175)      /* Create a text message. */
(176)      pTextMessage = SessionCreateTextMessage(pTopicSession, &iErr,
(177)      szErrBuf);
(178)      check_error(iErr, szErrBuf, 1);
(179)
(180)      /* Clear the body (payload) of the message. */
(181)      ClearBody(pTextMessage, &iErr, szErrBuf);/* set the mode to r/w */
(182)      check_error(iErr, szErrBuf, 1);
(183)
(184)      /* Copy in the text to be sent. */
(185)      SetText(pTextMessage, pBuffer, &iErr, szErrBuf);
(186)      check_error(iErr, szErrBuf, 1);
(187)
(188)      /* Set the JMSType of the message to "ASCII". */
(189)      SetJMSType(pTextMessage, (char*)"ASCII",&iErr, szErrBuf);
(190)      check_error(iErr, szErrBuf, 1);
```

```
(191)      printf("Sending Text Message: %s\n", pBuffer);
(192)
(193)      /* Publish the message. */
(194)      TopicPublisherPublish(pTopicPublisher, pTextMessage, &iErr,
(195)      szErrBuf);
(196)      check_error(iErr, szErrBuf, 1);
(197)
(198)      /* Commit the session. */
(199)      SessionCommit(pTopicSession, &iErr, szErrBuf);
(200)      check_error(iErr, szErrBuf, 1);
(201)
(202)      /* Close and clean up. */
(203)      TopicPublisherClose(pTopicPublisher, &iErr, szErrBuf);
(204)      check_error(iErr, szErrBuf, 1);
(205)      SessionClose(pTopicSession, &iErr, szErrBuf);
(206)      check_error(iErr, szErrBuf, 1);
(207)      ConnectionClose(pTopicConnection, &iErr, szErrBuf);
(208)      check_error(iErr, szErrBuf, 1);
(209)      DeleteMessage(pTextMessage, &iErr, szErrBuf);
(210)      DeleteTopicPublisher(pTopicPublisher, &iErr, szErrBuf);
(211)      DeleteSession(pTopicSession, &iErr, szErrBuf);
(212)      DeleteDestination(pTopic, &iErr, szErrBuf);
(213)      DeleteConnection(pTopicConnection, &iErr, szErrBuf);
(214)      DeleteTopicConnectionFactory(pTcf, &iErr, szErrBuf);
(215) }
(216)
(217)
(218) /*
(219)  * ============================================================
(220)  * Topic Subscriber
(221)  * This routine demonstrates how to subsribe a message from
(222)  * a topic.
(223)  * ============================================================
(224) */
(225) static void subscriber() {
(226)      SBYN_Session*           pTopicSession;
(227)      SBYN_Destination*       pTopic;
(228)      SBYN_Message*           pReceivedMessage = 0;
(229)      SBYN_TopicSubscriber*   pTopicSubscriber;
(230)      SBYN_Connection*        pTopicConnection;
(231)      SBYN_TopicConnectionFactory* pTcf;
(232)      unsigned long           ulMessageSize = 1024;
(233)      unsigned long           ulMessageCount = 10;
(234)      int                     iErr;
(235)      char                    szErrBuf[256];
(236)      char                    szUserInput[80];
(237)      char                    pTopicName[] = "eGatePubSubSample";
(238)
(239)      /* create a topic connection */
(240)      pTcf = CreateTopicConnectionFactory(spHostName, susPort, &iErr,
(241)      szErrBuf);
(242)      check_error(iErr, szErrBuf, 1);
(243)      if(!pTcf) {
(244)           printf("CreateTopicConnectionFactory
(245)      failed\nError:%0X\nReason:%s\n", iErr, szErrBuf);
(246)           exit(2);
(247)      }
(248)
(249)      /* create a topic connection */
(250)      pTopicConnection = CreateTopicConnection(pTcf,&iErr, szErrBuf);
(251)      check_error(iErr, szErrBuf, 1);
(252)
(253)      /* set client ID */
(254)      ConnectionSetClientID(pTopicConnection,
```

```
(255)      (char*)"TopicTestSubConnection",&iErr, szErrBuf);
(256)      check_error(iErr, szErrBuf, 1);
(257)
(258)      /* start connection */
(259)      ConnectionStart(pTopicConnection,&iErr, szErrBuf);
(260)      check_error(iErr, szErrBuf, 1);
(261)
(262)      /* create a topic session */
(263)      pTopicSession = ConnectionCreateTopicSession (pTopicConnection,
(264)      SBYN_TRANSACTED, SBYN_CLIENT_ACKNOWLEDGE, &iErr, szErrBuf);
(265)      check_error(iErr, szErrBuf, 1);
(266)      if(!pTopicSession) {
(267)           printf("CreateTopicSession failed\nError:%0X\nReason:%s\n",
(268)      iErr, szErrBuf);
(269)           exit(2);
(270)      }
(271)
(272)      /* create a topic */
(273)      pTopic = SessionCreateTopic(pTopicSession, pTopicName, &iErr,
(274)      szErrBuf);
(275)      check_error(iErr, szErrBuf, 1);
(276)
(277)      /* create a subscriber */
(278)      pTopicSubscriber = SessionCreateDurableSubscriber(pTopicSession,
(279)      pTopic, (char*)"TopicTestSubscriber",&iErr, szErrBuf);
(280)      check_error(iErr, szErrBuf, 1);
(281)
(282)      printf("Waiting for message ... \n");
(283)      do {
(284)           /* waiting for incoming messages */
(285)           pReceivedMessage = TopicSubscriberReceive(pTopicSubscriber,
(286)      &iErr, szErrBuf);
(287)           check_error(iErr, szErrBuf, 1);
(288)           if (pReceivedMessage->type == SBYN_MESSAGE_TYPE_TEXT){
(289)                char *pReturnedBuf;
(290)                SBYN_WString *pWstr;
(291)                char* pMessageType;
(292)                /* retrive the JMS message type */
(293)                pWstr = GetJMSType(pReceivedMessage, &iErr, szErrBuf);
(294)                check_error(iErr, szErrBuf, 1);
(295)                pMessageType = WStringToChar(pWstr);
(296)                check_error(iErr, szErrBuf, 1);
(297)
(298)                /* retrive the text from message */
(299)              pReturnedBuf = GetText(pReceivedMessage, &iErr, szErrBuf);
(300)                printf("Received text message (JMSType:%s):  %s\n",
(301)      pMessageType, pReturnedBuf);
(302)                free(pReturnedBuf);
(303)                DeleteWString(pWstr);
(304)                free((void*)pMessageType);
(305)            SessionCommit(pTopicSession, &iErr, szErrBuf);
(306)           }
(307)           printf("Enter 'r' for receiving more message, 'q' for
(308)      exit\n");
(309)           scanf("%s", szUserInput);
(310)      } while ( szUserInput[0] != 'q' && szUserInput[0] != 'Q' );
(311)
(312)      check_error(iErr, szErrBuf, 1);
(313)
(314)      /* close subscriber, session ... */
(315)      TopicSubscriberClose(pTopicSubscriber, &iErr, szErrBuf);
(316)      SessionClose(pTopicSession, &iErr, szErrBuf);
(317)      ConnectionClose(pTopicConnection, &iErr, szErrBuf);
(318)
```

```
(319)      /* delete objects */
(320)      DeleteMessage(pReceivedMessage, &iErr, szErrBuf);
(321)      DeleteDestination(pTopic, &iErr, szErrBuf);
(322)      DeleteTopicSubscriber(pTopicSubscriber, &iErr, szErrBuf);
(323)      DeleteSession(pTopicSession, &iErr, szErrBuf);
(324)      DeleteConnection(pTopicConnection, &iErr, szErrBuf);
(325)      DeleteTopicConnectionFactory(pTcf, &iErr, szErrBuf);
(326) }
```

7.2.2. Queue Messaging (Sending/Receiving) Using C

```
(1)   /* ------------------------------------------------------------------
(2)    *   Sample code to demonstrate JMS Queue Messaging using C.
(3)    *
(4)   * ------------------------------------------------------------------
(5)    *
(6)    *  Disclaimer:
(7)    *
(8)    *   Copyright 2002 by SeeBeyond Technology Corporation.
(9)    *   All Rights Reserved.
(10)   *
(11)   *   Unless otherwise stipulated in a written agreement for this software,
(12)   *   duly executed by SeeBeyond Technology Corporation, this software is
(13)   *   provided as is without warranty of any kind.  The entire risk as to
(14)   *      the results and performance of this software is assumed by the user.
(15)   *   SeeBeyond Technology Corporation disclaims all warranties, either
(16)   *   express or implied, including but not limited, the implied warranties
(17)   *   of merchantability, fitness for a particular purpose, title and
(18)   *   non-infringement, with respect to this software.
(19)   *
(20)   * ------------------------------------------------------------------*/
(21)
(22)  #include <mscapi.h>
(23)  #include <stdlib.h>
(24)  #include <stdio.h>
(25)  #include <string.h>
(26)  #ifndef WIN32
(27)  #include <unistd.h>
(28)  #endif
(29)
(30)  #if defined(OS400)
(31)  extern char *optarg;
(32)  #endif
(33)
(34)  #if defined(__gnu__)
(35)  #include <getopt.h>
(36)  #endif
(37)
(38)  char                        optionProducer[] = "[ -u ] [ -p port ]
(39)                              [ -h hostname ]";
(40)  char                        optionConsumer[] = "[ -c ] [ -p port ]
(41)                              [ -h hostname ]";
(42)  char                        optdescription[] = "\t-u run as a
(43)                              producer\n\t-c run as a consumer\n\t-p port
(44)                              number\n\t-h hostname\n";
(45)  char*                       spHostName;
(46)  char                        localHost[] = "localhost";
(47)  static unsigned short       susPort = 24053;
(48)  int                         iErr;
(49)  char                        szErrBuf[256];
(50)
(51)  /* Routine for checking errors.*/
(52)  static void check_error(int err, char* errBuf, int exitnow)
(53)  {
```

```
(54)       if (err){
(55)            printf("ERROR:0x%x - %s\n", err, errBuf);
(56)            if (exitnow)
(57)                 exit(1);
(58)       }
(59)  }
(60)
(61)  int main(int argc, char *argv[]) {
(62)       int            c;
(63)       char           cOption = 0;
(64)
(65)       spHostName = localHost;
(66)
(67)       while((c = getopt(argc, argv, ":p:h:P:H:cCUu")) != -1) {
(68)            switch(c){
(69)            case 'p':
(70)            case 'P':
(71)                 susPort = atoi(optarg);  /* setup the port number */
(72)                 break;
(73)            case 'h':
(74)            case 'H':
(75)                 spHostName = optarg;     /* setup the hostname */
(76)                 break;
(77)            case 'U':
(78)            case 'u':
(79)                 cOption = 'u';           /* run as a producer */
(80)                 break;
(81)            case 'c':
(82)            case 'C':
(83)                 cOption = 'c';           /* run as a consumer */
(84)                 break;
(85)            case ':':
(86)            case '?':
(87)                 printf("\nSYNOPSIS\n");
(88)                 printf("%s %s\n", argv[0], optionProducer);
(89)                 printf("%s %s\n", argv[0], optionConsumer);
(90)                 printf("%s\n", optdescription);
(91)                 exit(1);
(92)                 break;
(93)            }
(94)       }
(95)
(96)       if (cOption == 'u'){
(97)            sender();/* invoke producer */
(98)       } else if (cOption == 'c'){
(99)            receiver();/* invoke consumer */
(100)      } else {
(101)           printf("\nSYNOPSIS\n");
(102)           printf("%s %s\n", argv[0], optionProducer);
(103)           printf("%s %s\n", argv[0], optionConsumer);
(104)           printf("%s\n", optdescription);
(105)           exit(1);
(106)      }
(107) }
(108)
(109)
(110) void receiver(){
(111)      char                              pQueueName[] = "eGateP2PSample";
(112)      SBYN_QueueConnectionFactory*      pQcf = NULL;
(113)      SBYN_Connection*                  pQueueConnection = NULL;
(114)      SBYN_Session*                     pQueueSession = NULL;
(115)      SBYN_Destination*                 pQueue = NULL;
(116)      SBYN_QueueReceiver*               pQueueReceiver = NULL;
(117)      SBYN_Message*                     pMessage = NULL;
```

```
(118)      unsigned int                    iBufLen = 0;
(119)      char                            szUserInput[80];
(120)
(121)      printf("Queue name is %s\n", pQueueName);
(122)
(123)      /* Create a queue connection factory */
(124)      pQcf = CreateQueueConnectionFactory(spHostName, susPort, &iErr,
     szErrBuf);
(125)      check_error(iErr, szErrBuf, 1);
(126)      if(!pQcf) {
(127)   printf("CreateQueueConnectionFactory failed\nError:%0X\nReason:%s\n",
     iErr, szErrBuf);
(128)   exit(2);
(129)      }
(130)
(131)      /* Create a queue connection */
(132)      pQueueConnection = CreateQueueConnection(pQcf,&iErr, szErrBuf);
(133)      check_error(iErr, szErrBuf, 1);
(134)
(135)      /* Set the client ID */
(136)      ConnectionSetClientID(pQueueConnection, (char*)"RECEIVER", &iErr,
     szErrBuf);
(137)      check_error(iErr, szErrBuf, 1);
(138)
(139)      /* Start the connection */
(140)      ConnectionStart(pQueueConnection,&iErr, szErrBuf);
(141)      check_error(iErr, szErrBuf, 1);
(142)
(143)      /* Create a queue session */
(144)      pQueueSession = ConnectionCreateQueueSession (pQueueConnection,
     SBYN_TRANSACTED, SBYN_CLIENT_ACKNOWLEDGE, &iErr, szErrBuf);
(145)      check_error(iErr, szErrBuf, 1);
(146)      if(!pQueueSession) {
(147)   printf("CreateTopicSession failed\nError:%0X\nReason:%s\n", iErr,
     szErrBuf);
(148)   exit(2);
(149)      }
(150)
(151)      /* Create a queue */
(152)      pQueue = SessionCreateQueue(pQueueSession, pQueueName, &iErr,
     szErrBuf);
(153)      check_error(iErr, szErrBuf, 1);
(154)
(155)      /* Create a queue receiver */
(156)      pQueueReceiver = SessionCreateReceiver(pQueueSession, pQueue,
     &iErr, szErrBuf);
(157)      check_error(iErr, szErrBuf, 1);
(158)
(159)      do {
(160)           /* Blocking for the message */
(161)   pMessage = QueueReceiverReceive(pQueueReceiver, &iErr, szErrBuf);
(162)   check_error(iErr, szErrBuf, 1);
(163)   if (pMessage->type == SBYN_MESSAGE_TYPE_TEXT){
(164)        char *rtbuf;
(165)        rtbuf = GetText(pMessage, &iErr, szErrBuf);
(166)        printf("Received message:  %s\n", rtbuf);
(167)        free(rtbuf);
(168)   }
(169)   printf("Enter 'r' for receiving more message, 'q' for exit\n");
(170)   scanf("%s", szUserInput);
(171)      } while ( szUserInput[0] != 'q' && szUserInput[0] != 'Q' );
(172)
(173)      /* now close the connections */
(174)      QueueReceiverClose(pQueueReceiver, &iErr, szErrBuf);
```

```
(175)      check_error(iErr, szErrBuf, 1);
(176)      SessionClose(pQueueSession, &iErr, szErrBuf);
(177)      check_error(iErr, szErrBuf, 1);
(178)      ConnectionStop(pQueueConnection, &iErr, szErrBuf);
(179)      check_error(iErr, szErrBuf, 1);
(180)      ConnectionClose(pQueueConnection, &iErr, szErrBuf);
(181)      check_error(iErr, szErrBuf, 1);
(182)
(183)      /* delete the objects */
(184)      DeleteMessage(pMessage, &iErr, szErrBuf);
(185)      DeleteQueueReceiver(pQueueReceiver, &iErr, szErrBuf);
(186)      DeleteDestination(pQueue, &iErr, szErrBuf);
(187)      DeleteSession(pQueueSession, &iErr, szErrBuf);
(188)      DeleteConnection(pQueueConnection, &iErr, szErrBuf);
(189)      DeleteQueueConnectionFactory(pQcf, &iErr, szErrBuf);
(190) }
(191)
(192)
(193)
(194) void sender(){
(195)      char                          pQueueName[] = "P2PSample";
(196)      SBYN_QueueConnectionFactory* pQcf = NULL;
(197)      SBYN_Connection*              pQueueConnection = NULL;
(198)      SBYN_Session*                 pQueueSession = NULL;
(199)      SBYN_Destination*             pQueue = NULL;
(200)      SBYN_QueueSender*             pQueueSender = NULL;
(201)      SBYN_Message*                 textMessage = NULL;
(202)      const int                     MAX_MESSAGE_SIZE = 60;
(203)      char                          pBuffer[] = "This is a text message";
(204)
(205)      /* Create a queue connection factory */
(206)      pQcf = CreateQueueConnectionFactory(spHostName, susPort, &iErr,
     szErrBuf);
(207)      check_error(iErr, szErrBuf, 1);
(208)      if(!pQcf) {
(209)   printf("CreateQueueConnectionFactory failed\nError:%0X\nReason:%s\n",
     iErr, szErrBuf);
(210)   exit(2);
(211)      }
(212)
(213)      /* Create a queue connection */
(214)      pQueueConnection = CreateQueueConnection(pQcf,&iErr, szErrBuf);
(215)      check_error(iErr, szErrBuf, 1);
(216)
(217)      /* Set the client ID */
(218)      ConnectionSetClientID(pQueueConnection, (char*)"SENDER", &iErr,
     szErrBuf);
(219)      check_error(iErr, szErrBuf, 1);
(220)
(221)      /* Start the connection */
(222)      ConnectionStart(pQueueConnection,&iErr, szErrBuf);
(223)      check_error(iErr, szErrBuf, 1);
(224)
(225)      /* Create a queue session */
(226)      pQueueSession = ConnectionCreateQueueSession (pQueueConnection,
     SBYN_TRANSACTED, SBYN_CLIENT_ACKNOWLEDGE, &iErr, szErrBuf);
(227)      check_error(iErr, szErrBuf, 1);
(228)      if(!pQueueSession) {
(229)   printf("CreateTopicSession failed\nError:%0X\nReason:%s\n", iErr,
     szErrBuf);
(230)   exit(2);
(231)      }
(232)
(233)      /* Create a queue */
```

```
(234)      pQueue = SessionCreateQueue(pQueueSession, pQueueName, &iErr,
     szErrBuf);
(235)      check_error(iErr, szErrBuf, 1);
(236)
(237)      /* Create a queue sender */
(238)      pQueueSender = SessionCreateSender(pQueueSession, pQueue, &iErr,
     szErrBuf);
(239)      check_error(iErr, szErrBuf, 1);
(240)
(241)      /* Create a text message */
(242)      textMessage = SessionCreateTextMessage(pQueueSession, &iErr,
     szErrBuf);
(243)      check_error(iErr, szErrBuf, 1);
(244)
(245)      /* set the mode to r/w */
(246)      ClearBody(textMessage, &iErr, szErrBuf);
(247)      check_error(iErr, szErrBuf, 1);
(248)
(249)      /* put in text */
(250)      SetText(textMessage, pBuffer, &iErr, szErrBuf);
(251)      check_error(iErr, szErrBuf, 1);
(252)      printf("Sending Message %s\n", pBuffer);
(253)
(254)      /* send out the message */
(255)      QueueSenderSend(pQueueSender, textMessage, &iErr, szErrBuf);
(256)      check_error(iErr, szErrBuf, 1);
(257)
(258)      /* session commit */
(259)      SessionCommit(pQueueSession, &iErr, szErrBuf);
(260)      check_error(iErr, szErrBuf, 1);
(261)
(262)      /* now close the connections */
(263)      QueueSenderClose(pQueueSender, &iErr, szErrBuf);
(264)      check_error(iErr, szErrBuf, 1);
(265)      SessionClose(pQueueSession, &iErr, szErrBuf);
(266)      check_error(iErr, szErrBuf, 1);
(267)      ConnectionStop(pQueueConnection, &iErr, szErrBuf);
(268)      check_error(iErr, szErrBuf, 1);
(269)      ConnectionClose(pQueueConnection, &iErr, szErrBuf);
(270)      check_error(iErr, szErrBuf, 1);
(271)
(272)      /* delete the objects */
(273)      DeleteMessage(textMessage, &iErr, szErrBuf);
(274)      DeleteQueueSender(pQueueSender, &iErr, szErrBuf);
(275)      DeleteDestination(pQueue, &iErr, szErrBuf);
(276)      DeleteSession(pQueueSession, &iErr, szErrBuf);
(277)      DeleteConnection(pQueueConnection, &iErr, szErrBuf);
(278)      DeleteQueueConnectionFactory(pQcf, &iErr, szErrBuf);
(279) }
```

7.2.3. Request-Reply Messaging Using C

```
(1)   /* ------------------------------------------------------------------
(2)    *   Sample code to demonstrate JMS Request-Reply messaging using C.
(3)    *
(4)   * ------------------------------------------------------------------
(5)    *
(6)    *  Disclaimer:
(7)    *
(8)    *   Copyright 2002 by SeeBeyond Technology Corporation.
(9)    *   All Rights Reserved.
(10)   *
(11)   *   Unless otherwise stipulated in a written agreement for this software,
(12)   *   duly executed by SeeBeyond Technology Corporation, this software is
```

```
(13)   *   provided as is without warranty of any kind.  The entire risk as to
(14)   *      the results and performance of this software is assumed by the user.
(15)   *   SeeBeyond Technology Corporation disclaims all warranties, either
(16)   *   express or implied, including but not limited, the implied warranties
(17)   *   of merchantability, fitness for a particular purpose, title and
(18)   *   non-infringement, with respect to this software.
(19)   *
(20)   * ----------------------------------------------------------------------*/
(21)
(22)  #include "mscapi.h"
(23)  #include <stdlib.h>
(24)  #include <stdio.h>
(25)  #include <string.h>
(26)
(27)  #ifndef WIN32
(28)  #include <unistd.h>
(29)  #endif
(30)  #if defined(WIN32)
(31)  #include <sbyn_getopt.h>
(32)  #endif
(33)
(34)  #if defined(OS400)
(35)  extern char *optarg;
(36)  #endif
(37)
(38)  #if defined(__gnu__)
(39)  #include <getopt.h>
(40)  #endif
(41)
(42)  static void requestor();
(43)
(44)  /* Routine for checking errors.*/
(45)  static void check_error(int err, char* errBuf, int exitnow)
(46)  {
(47)       if (err){
(48)            printf("ERROR:0x%x - %s\n", err, errBuf);
(49)            if (exitnow)
(50)                 exit(1);
(51)       }
(52)  }
(53)
(54)  char                    optionRequestor[] = "[ -r ] [ -p port ]
(55)                          [ -h hostname ]";
(56)  char                    optdescription[] = "\t-r run as a
(57)                          requestor\n\t-p port number\n\t-h
(58)                          hostname\n";
(59)  char*                   spHostName;
(60)  char                    localHost[] = "localhost";
(61)  static unsigned short   susPort = 24053;
(62)  int                     iErr;
(63)  char                    szErrBuf[256];
(64)  char                    pQueueName[] = "QueueRequestorSample";
(65)
(66)  int main(int argc, char *argv[]) {
(67)       int           c;
(68)       char          cOption = 0;
(69)
(70)       spHostName = localHost;
(71)
(72)       while((c = getopt(argc, argv, ":p:h:P:H:rR")) != -1) {
(73)            switch(c){
(74)            case 'p':
(75)            case 'P':
(76)                 susPort = atoi(optarg);  /* setup the port number */
(77)                 break;
```

```
(78)            case 'h':
(79)            case 'H':
(80)                 spHostName = optarg;     /* setup the hostname */
(81)                 break;
(82)            case 'R':
(83)            case 'r':
(84)                 cOption = 'r';           /* run as a requestor */
(85)                 break;
(86)            case ':':
(87)            case '?':
(88)                 printf("\nSYNOPSIS\n");
(89)                 printf("%s %s\n", argv[0], optionRequestor);
(90)                 printf("%s\n", optdescription);
(91)                 exit(1);
(92)             }
(93)       }
(94)
(95)       if (cOption == 'r'){
(96)            requestor();/* invoke requestor */
(97)       } else {
(98)            printf("\nSYNOPSIS\n");
(99)            printf("%s %s\n", argv[0], optionRequestor);
(100)           printf("%s\n", optdescription);
(101)           exit(1);
(102)      }
(103) }
(104)
(105)
(106)
(107) /*
(108)  * ============================================================
(109)  * Queue Requestor
(110)  *   This routine demonstrates how to do request/reply.
(111)  * ============================================================
(112) */
(113) void requestor(){
(114)      SBYN_QueueConnectionFactory *pQcf     = NULL;
(115)      SBYN_Connection *pQueueConnection     = NULL;
(116)      SBYN_Session *pQueueSession           = NULL;
(117)      SBYN_Destination *pQueue              = NULL;
(118)      SBYN_QueueSender *pQueueSender        = NULL;
(119)      SBYN_Message *textMessage             = NULL;
(120)      SBYN_Message *pReplyMessage           = NULL;
(121)      SBYN_QueueRequestor *pQueueRequestor  = 0;
(122)      char pBuffer[] = "This is a text message";
(123)
(124)      /* Create a queue connection factory */
(125)      pQcf = CreateQueueConnectionFactory(spHostName, susPort, &iErr,
      szErrBuf);
(126)      check_error(iErr, szErrBuf, 1);
(127)      if(!pQcf) {
(128)           printf("CreateQueueConnectionFactory
      failed\nError:%0X\nReason:%s\n", iErr, szErrBuf);
(129)           exit(2);
(130)      }
(131)
(132)      /* Create a queue connection */
(133)      pQueueConnection = CreateQueueConnection(pQcf,&iErr, szErrBuf);
(134)      check_error(iErr, szErrBuf, 1);
(135)
(136)      /* Set the client ID */
(137)      ConnectionSetClientID(pQueueConnection, (char*)"REQUESTOR",&iErr,
      szErrBuf);
(138)      check_error(iErr, szErrBuf, 1);
```

```
(139)
(140)      /* Start the connection */
(141)      ConnectionStart(pQueueConnection,&iErr, szErrBuf);
(142)      check_error(iErr, szErrBuf, 1);
(143)
(144)      /* Create a queue session */
(145)      pQueueSession = ConnectionCreateQueueSession (pQueueConnection,
      SBYN_NON_TRANSACTED, SBYN_CLIENT_ACKNOWLEDGE, &iErr, szErrBuf);
(146)      check_error(iErr, szErrBuf, 1);
(147)      if(!pQueueSession) {
(148)           printf("CreateTopicSession failed\nError:%0X\nReason:%s\n",
      iErr, szErrBuf);
(149)           exit(2);
(150)      }
(151)
(152)      /* Create a queue */
(153)      pQueue = SessionCreateQueue(pQueueSession, pQueueName, &iErr,
      szErrBuf);
(154)      check_error(iErr, szErrBuf, 1);
(155)
(156)      /* Create a queue requestor */
(157)      pQueueRequestor = CreateQueueRequestor(pQueueSession,pQueue,
      &iErr, szErrBuf);
(158)      check_error(iErr, szErrBuf, 1);
(159)
(160)      /* Create a text message and make a request */
(161)      textMessage = SessionCreateTextMessage(pQueueSession, &iErr,
      szErrBuf);
(162)      check_error(iErr, szErrBuf, 1);
(163)
(164)      /* set the mode to r/w */
(165)      ClearBody(textMessage, &iErr, szErrBuf);
(166)      check_error(iErr, szErrBuf, 1);
(167)
(168)      /* Copy in the text to be sent. */
(169)      SetText(textMessage, pBuffer, &iErr, szErrBuf);
(170)      check_error(iErr, szErrBuf, 1);
(171)      printf("Sending Message: %s\n", pBuffer);
(172)
(173)      /* Set ReplyTo destination */
(174)      SetJMSReplyTo(textMessage, pQueue, &iErr, szErrBuf);
(175)      check_error(iErr, szErrBuf, 1);
(176)
(177)      /* Make a request and wait for a reply */
(178)      pReplyMessage = QueueRequestorRequestTimeOut(pQueueRequestor,
      textMessage, 100000, &iErr, szErrBuf);
(179)      check_error(iErr, szErrBuf, 1);
(180)
(181)      /* Extract the message type */
(182)      if (GetMessageType(pReplyMessage, &iErr, szErrBuf) ==
      SBYN_MESSAGE_TYPE_TEXT){
(183)           char *rtbuf;
(184)   check_error(iErr, szErrBuf, 1);
(185)   /* Extract the text */
(186)           rtbuf = GetText(pReplyMessage, &iErr, szErrBuf);
(187)   check_error(iErr, szErrBuf, 1);
(188)           printf("Received message:  %s\n", rtbuf);
(189)           free(rtbuf);
(190)      }
(191)      DeleteMessage(pReplyMessage, &iErr, szErrBuf);
(192)      check_error(iErr, szErrBuf, 1);
(193)
(194)      /* now close the connections */
(195)      SessionClose(pQueueSession, &iErr, szErrBuf);
```

```
(196)      check_error(iErr, szErrBuf, 1);
(197)      ConnectionStop(pQueueConnection, &iErr, szErrBuf);
(198)      check_error(iErr, szErrBuf, 1);
(199)      ConnectionClose(pQueueConnection, &iErr, szErrBuf);
(200)      check_error(iErr, szErrBuf, 1);
(201)
(202)      /* delete the objects */
(203)      DeleteMessage(textMessage, &iErr, szErrBuf);
(204)      DeleteQueueSender(pQueueSender, &iErr, szErrBuf);
(205)      DeleteDestination(pQueue, &iErr, szErrBuf);
(206)      DeleteSession(pQueueSession, &iErr, szErrBuf);
(207)      DeleteConnection(pQueueConnection, &iErr, szErrBuf);
(208)      DeleteQueueConnectionFactory(pQcf, &iErr, szErrBuf);
(209) }
(210)
(211)
(212)
(213) void receiver(){
(214)      SBYN_QueueConnectionFactory* pQcf = NULL;
(215)      SBYN_Connection* pQueueConnection = NULL;
(216)      SBYN_Session* pQueueSession = NULL;
(217)      SBYN_Destination* pQueue = NULL;
(218)      SBYN_Destination* pReplyToQueue = NULL;
(219)      SBYN_QueueReceiver* pQueueReceiver = NULL;
(220)      SBYN_QueueSender* pReplyQueueSender = NULL;
(221)      SBYN_Message *pMessage = NULL;
(222)      char szUserInput[80];
(223)
(224)      printf("Queue name is %s\n", pQueueName);
(225)      pQcf = CreateQueueConnectionFactory(spHostName, susPort, &iErr,
     szErrBuf);
(226)      if(!pQcf) {
(227)           printf("CreateQueueConnectionFactory
     failed\nError:%0X\nReason:%s\n", iErr, szErrBuf);
(228)           exit(2);
(229)      }
(230)
(231)      pQueueConnection = CreateQueueConnection(pQcf,&iErr, szErrBuf);
(232)      ConnectionSetClientID(pQueueConnection, (char*)"RECEIVER",&iErr,
     szErrBuf);
(233)      ConnectionStart(pQueueConnection,&iErr, szErrBuf);
(234)      pQueueSession = ConnectionCreateQueueSession (pQueueConnection,
     SBYN_TRANSACTED, SBYN_CLIENT_ACKNOWLEDGE, &iErr, szErrBuf);
(235)      if(!pQueueSession) {
(236)           printf("CreateTopicSession failed\nError:%0X\nReason:%s\n",
     iErr, szErrBuf);
(237)           exit(2);
(238)      }
(239)      pQueue = SessionCreateQueue(pQueueSession, pQueueName, &iErr,
     szErrBuf);
(240)      pQueueReceiver = SessionCreateReceiver(pQueueSession, pQueue,
     &iErr, szErrBuf);
(241)
(242)      do {
(243)           pMessage = QueueReceiverReceive(pQueueReceiver, &iErr,
     szErrBuf);
(244)           if (pMessage->type == SBYN_MESSAGE_TYPE_TEXT){
(245)                char *rtbuf;
(246)                rtbuf = GetText(pMessage, &iErr, szErrBuf);
(247)                printf("Reading message:  %s\n", rtbuf);
(248)                free(rtbuf);
(249)           } else {
(250)                printf("Error: Received invalid message format\n");
(251)           }
```

```
(252)           pReplyToQueue = GetJMSReplyTo(pMessage, &iErr, szErrBuf);
(253)           pReplyQueueSender = SessionCreateSender(pQueueSession,
     pReplyToQueue, &iErr, szErrBuf);
(254)           ClearBody(pMessage, &iErr, szErrBuf);
(255)           SetText(pMessage, (char*)"This is reply message", &iErr,
     szErrBuf);
(256)          QueueSenderSend(pReplyQueueSender, pMessage, &iErr, szErrBuf);
(257)           SessionCommit(pQueueSession, &iErr, szErrBuf);
(258)           printf("Enter 'r' for receiving more message, 'q' for
     exit\n");
(259)           scanf("%s", szUserInput);
(260)      } while ( szUserInput[0] != 'q' && szUserInput[0] != 'Q' );
(261)
(262)      /* now close the connections */
(263)      QueueReceiverClose(pQueueReceiver, &iErr, szErrBuf);
(264)      SessionClose(pQueueSession, &iErr, szErrBuf);
(265)      ConnectionStop(pQueueConnection, &iErr, szErrBuf);
(266)      ConnectionClose(pQueueConnection, &iErr, szErrBuf);
(267)
(268)      /* delete the objects */
(269)      DeleteMessage(pMessage, &iErr, szErrBuf);
(270)      DeleteQueueReceiver(pQueueReceiver, &iErr, szErrBuf);
(271)      DeleteDestination(pQueue, &iErr, szErrBuf);
(272)      DeleteSession(pQueueSession, &iErr, szErrBuf);
(273)      DeleteConnection(pQueueConnection, &iErr, szErrBuf);
(274)      DeleteQueueConnectionFactory(pQcf, &iErr, szErrBuf);
(275) }
```

7.2.4. Message Selector Using C

```
(1)   /* ------------------------------------------------------------------
(2)    *   Sample code to demonstrate JMS Message Selectors using C.
(3)    *
(4)   * ------------------------------------------------------------------
(5)    *
(6)    *  Disclaimer:
(7)    *
(8)    *   Copyright 2002 by SeeBeyond Technology Corporation.
(9)    *   All Rights Reserved.
(10)   *
(11)   *   Unless otherwise stipulated in a written agreement for this software,
(12)   *   duly executed by SeeBeyond Technology Corporation, this software is
(13)   *   provided as is without warranty of any kind.  The entire risk as to
(14)   *      the results and performance of this software is assumed by the user.
(15)   *   SeeBeyond Technology Corporation disclaims all warranties, either
(16)   *   express or implied, including but not limited, the implied warranties
(17)   *   of merchantability, fitness for a particular purpose, title and
(18)   *   non-infringement, with respect to this software.
(19)   *
(20)   * ------------------------------------------------------------------*/
(21)
(22)  #include <mscapi.h>
(23)  #include <stdlib.h>
(24)  #include <stdio.h>
(25)  #include <string.h>
(26)
(27)  #ifndef WIN32
(28)  #include <unistd.h>
(29)  #endif
(30)  #if defined(WIN32)
(31)  #include <sbyn_getopt.h>
(32)  #endif
(33)
(34)  #if defined(OS400)
```

```
(35)  extern char *optarg;
(36)  #endif
(37)
(38)  #if defined(__gnu__)
(39)  #include <getopt.h>
(40)  #endif
(41)
(42)  char                    optionProducer[] = "[ -u ] [ -p port ]
(43)                          [ -h hostname ]";
(44)  char                    optionConsumer[] = "[ -c ] [ -p port ]
(45)                          [ -h hostname ]";
(46)  char                    optdescription[] = "\t-u run as a
(47)                          producer\n\t-c run as a consumer\n\t-p
(48)                          port number\n\t-h hostname\n";
(49)  static char             localHost[] = "localhost";
(50)  static unsigned short   susPort = 24053; /* default port number */
(51)  unsigned long           sulMessageSize = 16; /* default host name */
(52)  static char*            spHostName;
(53)  static char             PROP_NAME[] = "property";
(54)  int                     iErr;
(55)  char                    szErrBuf[256];
(56)
(57)  static void selector_publisher();
(58)  static void selector_subscriber();
(59)
(60)  /* Check for errors. */
(61)  static void check_error(int err, char* errBuf, int exitnow)
(62)  {
(63)       if (err){
(64)            printf("ERROR:0x%x - %s\n", err, errBuf);
(65)            if (exitnow)
(66)                 exit(1);
(67)       }
(68)  }
(69)
(70)
(71)  int main(int argc, char *argv[]) {
(72)       int           c;
(73)       char          cOption = 0;
(74)
(75)       spHostName = localHost;
(76)
(77)       while((c = getopt(argc, argv, ":p:h:P:H:cCUu")) != -1) {
(78)            switch(c){
(79)            case 'p':
(80)            case 'P':
(81)                 susPort = atoi(optarg);  /* setup the port number */
(82)                 break;
(83)            case 'h':
(84)            case 'H':
(85)                 spHostName = optarg;     /* setup the hostname */
(86)                 break;
(87)            case 'U':
(88)            case 'u':
(89)                 cOption = 'u';           /* run as a producer */
(90)                 break;
(91)            case 'c':
(92)            case 'C':
(93)                 cOption = 'c';           /* run as a consumer */
(94)                 break;
(95)            case ':':
(96)            case '?':
(97)                 printf("\nSYNOPSIS\n");
(98)                 printf("%s %s\n", argv[0], optionProducer);
```

```
(99)                printf("%s %s\n", argv[0], optionConsumer);
(100)               printf("%s\n", optdescription);
(101)               exit(1);
(102)               break;
(103)          }
(104)     }
(105)
(106)     if (cOption == 'u'){
(107)          selector_publisher();          /* invoke producer */
(108)     } else if (cOption == 'c'){
(109)          selector_subscriber();         /* invoke consumer */
(110)     } else {
(111)          printf("\nSYNOPSIS\n");
(112)          printf("%s %s\n", argv[0], optionProducer);
(113)          printf("%s %s\n", argv[0], optionConsumer);
(114)          printf("%s\n", optdescription);
(115)          exit(1);
(116)     }
(117) }
(118)
(119) static void selector_publisher(){
(120)     SBYN_TopicConnectionFactory*        pTcf;
(121)     SBYN_Connection*                    pTopicConnection = NULL;
(122)     SBYN_Session*                       pTopicSession = NULL;
(123)     SBYN_Destination*                   pTopic = NULL;
(124)     SBYN_TopicPublisher*                pTopicPublisher = NULL;
(125)     int                                 ii;
(126)     SBYN_Message                        msglist[10];
(127)     static char                         TOPIC_NAME[] = "Selector";
(128)
(129)     /* Create a topic factory. */
(130)     pTcf = CreateTopicConnectionFactory(spHostName, susPort, &iErr,
     szErrBuf);
(131)     if(!pTcf) {
(132)          printf("CreateTopicConnectionFactory
     failed\nError:%0X\nReason:%s\n", iErr, szErrBuf);
(133)          exit(2);
(134)     }
(135)
(136)     /* Create a topic connection. */
(137)     pTopicConnection = CreateTopicConnection(pTcf,&iErr, szErrBuf);
(138)     check_error(iErr, szErrBuf, 1);
(139)
(140)     /* Set the client ID. */
(141)     ConnectionSetClientID(pTopicConnection, (char*)"Publisher",&iErr,
     szErrBuf);
(142)     check_error(iErr, szErrBuf, 1);
(143)
(144)     /* Start the connection. */
(145)     ConnectionStart(pTopicConnection, &iErr, szErrBuf);
(146)     check_error(iErr, szErrBuf, 1);
(147)
(148)     /* Create a topic session. */
(149)     pTopicSession = ConnectionCreateTopicSession (pTopicConnection,
     SBYN_TRANSACTED, SBYN_CLIENT_ACKNOWLEDGE, &iErr, szErrBuf);
(150)     check_error(iErr, szErrBuf, 1);
(151)     if(!pTopicSession) {
(152)          printf("CreateTopicSession failed\nError:%0X\nReason:%s\n",
     iErr, szErrBuf);
(153)          exit(2);
(154)     }
(155)
(156)     /* Create a topic. */
```

```
(157)      pTopic = SessionCreateTopic(pTopicSession, TOPIC_NAME, &iErr,
      szErrBuf);
(158)      check_error(iErr, szErrBuf, 1);
(159)
(160)      /* Create a topic publisher. */
(161)      pTopicPublisher = SessionCreatePublisher(pTopicSession, pTopic,
      &iErr, szErrBuf);
(162)      check_error(iErr, szErrBuf, 1);
(163)
(164)      /* Set delivery mode as persistent */
(165)      TopicPublisherSetDeliveryMode(pTopicPublisher, SBYN_PERSISTENT,
      &iErr, szErrBuf);
(166)      check_error(iErr, szErrBuf, 1);
(167)
(168)      /* publish 10 messages to the topic */
(169)      for(ii=0; ii<10 ;ii++){
(170)           int index;
(171)           char buf[80];
(172)           /* Create a text message. */
(173)          msglist[ii].message = SessionCreateTextMessage(pTopicSession,
      &iErr, szErrBuf);
(174)    /* Clear the body (payload) of the message. */
(175)           ClearBody((SBYN_Message*)msglist[ii].message, &iErr,
      szErrBuf);
(176)           check_error(iErr, szErrBuf, 1);
(177)           msglist[ii].type = SBYN_MESSAGE_TYPE_TEXT;
(178)           index = ii%10;
(179)           sprintf(buf, "%d", index);
(180)           /* Set the string property */
(181)           SetStringProperty((SBYN_Message*)msglist[ii].message,
      PROP_NAME, buf, &iErr, szErrBuf);
(182)           check_error(iErr, szErrBuf, 1);
(183)           /* Copy in the text to be sent. */
(184)          SetText((SBYN_Message*)msglist[ii].message, (char*)"This is a
      text message", &iErr, szErrBuf);
(185)           check_error(iErr, szErrBuf, 1);
(186)           /* Publish the message. */
(187)           TopicPublisherPublish(pTopicPublisher,
      (SBYN_Message*)msglist[ii].message, &iErr, szErrBuf);
(188)           check_error(iErr, szErrBuf, 1);
(189)           printf("... Published 1 message with property %s = %d\n",
      PROP_NAME, ii);
(190)      }
(191)      /* Commit the session. */
(192)      SessionCommit(pTopicSession,  &iErr, szErrBuf);
(193)      check_error(iErr, szErrBuf, 1);
(194)
(195)      /* close and delete objects */
(196)      TopicPublisherClose(pTopicPublisher, &iErr, szErrBuf);
(197)      check_error(iErr, szErrBuf, 1);
(198)      SessionClose(pTopicSession, &iErr, szErrBuf);
(199)      check_error(iErr, szErrBuf, 1);
(200)      ConnectionClose(pTopicConnection, &iErr, szErrBuf);
(201)      check_error(iErr, szErrBuf, 1);
(202)      DeleteTopicPublisher(pTopicPublisher, &iErr, szErrBuf);
(203)      DeleteSession(pTopicSession, &iErr, szErrBuf);
(204)      DeleteConnection(pTopicConnection, &iErr, szErrBuf);
(205)      DeleteTopicConnectionFactory(pTcf, &iErr, szErrBuf);
(206) }
(207)
(208)
(209) static void selector_subscriber(){
(210)      SBYN_TopicConnectionFactory *pTcf;
(211)      SBYN_Connection *pTopicConnection = 0;
```

```
(212)      SBYN_Session *pTopicSession = 0;
(213)      SBYN_Destination *topic = 0;
(214)      SBYN_TopicSubscriber *pTopicSubscriber = 0;
(215)      SBYN_Message *pMessage = 0;
(216)      char selectorString[80];
(217)      char selectorSubscriberName[80];
(218)      int selector = 7;
(219)      char* selectorName;
(220)      static char TOPIC_NAME[] = "eGateSelector";
(221)
(222)
(223)      /* create a topic connection */
(224)      pTcf = CreateTopicConnectionFactory(spHostName, susPort, &iErr,
      szErrBuf);
(225)      check_error(iErr, szErrBuf, 1);
(226)      if(!pTcf) {
(227)           printf("CreateTopicConnectionFactory
      failed\nError:%0X\nReason:%s\n", iErr, szErrBuf);
(228)           exit(2);
(229)      }
(230)
(231)      /* create a topic connection */
(232)      pTopicConnection = CreateTopicConnection(pTcf,&iErr, szErrBuf);
(233)      check_error(iErr, szErrBuf, 1);
(234)
(235)      /* set client ID */
(236)     ConnectionSetClientID(pTopicConnection, (char*)"Publisher", &iErr,
      szErrBuf);
(237)      check_error(iErr, szErrBuf, 1);
(238)
(239)      /* start connection */
(240)      ConnectionStart(pTopicConnection,&iErr, szErrBuf);
(241)      check_error(iErr, szErrBuf, 1);
(242)
(243)      /* create a topic session */
(244)      pTopicSession = ConnectionCreateTopicSession (pTopicConnection,
      SBYN_TRANSACTED, SBYN_CLIENT_ACKNOWLEDGE, &iErr, szErrBuf);
(245)      check_error(iErr, szErrBuf, 1);
(246)      if(!pTopicSession) {
(247)           printf("CreateTopicSession failed\nError:%0X\nReason:%s\n",
      iErr, szErrBuf);
(248)           exit(2);
(249)      }
(250)
(251)      /* create a topic */
(252)      topic = SessionCreateTopic(pTopicSession, TOPIC_NAME, &iErr,
      szErrBuf);
(253)      check_error(iErr, szErrBuf, 1);
(254)
(255)      /* create subscriber  with selector*/
(256)      sprintf(selectorString, "%s = '%d'", PROP_NAME, selector);
(257)      selectorString[strlen(selectorString)] = '\0';
(258)      sprintf(selectorSubscriberName, "SelectorSubscriber%d", selector);
(259)      pTopicSubscriber =
      SessionCreateDurableSubscriberMessageSelector(pTopicSession, topic,
      selectorSubscriberName, selectorString, 0, &iErr, szErrBuf);
(260)      check_error(iErr, szErrBuf, 1);
(261)
(262)      /* Get message using selector */
(263)     selectorName = TopicSubscriberGetMessageSelector(pTopicSubscriber,
      &iErr, szErrBuf);
(264)      check_error(iErr, szErrBuf, 1);
(265)      printf("using selector: %s\n", selectorName);
```

```
(266)      for (pMessage = TopicSubscriberReceive(pTopicSubscriber, &iErr,
     szErrBuf);
(267)           pMessage != 0;
(268)           pMessage = TopicSubscriberReceiveTimeOut(pTopicSubscriber,
     1000, &iErr, szErrBuf))
(269)      {
(270)           char* property = WStringToChar(GetStringProperty(pMessage,
     PROP_NAME, &iErr, szErrBuf));
(271)           printf("Received 1 message with %s = %s\n", PROP_NAME,
     property);
(272)      }
(273)      check_error(iErr, szErrBuf, 1);
(274)
(275)      /* Session commit */
(276)      SessionCommit(pTopicSession, &iErr, szErrBuf);
(277)      check_error(iErr, szErrBuf, 1);
(278)
(279)      /* close and delete objects */
(280)      TopicSubscriberClose(pTopicSubscriber, &iErr, szErrBuf);
(281)      check_error(iErr, szErrBuf, 1);
(282)      SessionClose(pTopicSession, &iErr, szErrBuf);
(283)      check_error(iErr, szErrBuf, 1);
(284)      ConnectionClose(pTopicConnection, &iErr, szErrBuf);
(285)      check_error(iErr, szErrBuf, 1);
(286)
(287)      DeleteTopicSubscriber(pTopicSubscriber, &iErr, szErrBuf);
(288)      DeleteSession(pTopicSession, &iErr, szErrBuf);
(289)      DeleteConnection(pTopicConnection, &iErr, szErrBuf);
(290)      DeleteTopicConnectionFactory(pTcf, &iErr, szErrBuf);
(291) }
```

7.2.5. Publish/Subscribe Messaging Using XA in C

```
(1)    *-------------------------------------------------------------------
(2)    * Sample code to demonstrate JMS Pub/Sub using XA.
(3)    *-------------------------------------------------------------------
(4)    *
(5)    *  Disclaimer:
(6)    *
(7)    *  Copyright 2002 by SeeBeyond Technology Corporation.
(8)    *  All Rights Reserved.
(9)    *   Unless otherwise stipulated in a written agreement for this
(10)   *  software, duly executed by SeeBeyond Technology Corporation, this
(11)   *  software is provided as is without warranty of any kind.  The
(12)   *  entire risk as to the results and performance of this software
(13)   *  is assumed by the user.SeeBeyond Technology Corporation disclaims
(14)   *  all warranties, either express or implied, including but not
(15)   *  limited, the implied warranties of merchantability, fitness for a
(16)   *  particular purpose, title and non-infringement, with respect to
(17)   *  this software.
(18)   *
(19)   * -------------------------------------------------------------------*/
(20)  #include "mscapi.h"
(21)  #include <stdlib.h>
(22)  #include <stdio.h>
(23)  #include <string.h>
(24)
(25)  #ifndef WIN32
(26)  #include <unistd.h>
(27)  #endif
(28)  #if defined(WIN32)
(29)  #include "sbyn_getopt.h"
(30)  #endif
(31)
```

```
(32)  #if defined(OS400)
(33)  extern char *optarg;
(34)  #endif
(35)
(36)  #if defined(__gnu__)
(37)  #include <getopt.h>
(38)  #endif
(39)
(40)
(41)  char                    optionProducer[] = "[ -u ] [ -p port ]
(42)                          [ -h hostname ]";
(43)  char                    optionConsumer[] = "[ -c ] [ -p port ]
(44)                          [ -h hostname ]";
(45)  char                    optdescription[] = "\t-u  run as a
(46)                          producer\n\t-c run as a consumer\n\t-p port
(47)                          number\n\t-h  hostname\n";
(48)  static char             localHost[] = "localhost";
(49)  static unsigned short   susPort = 24053; /* default port number */
(50)  unsigned long           sulMessageSize = 16; /* default host name */
(51)  static char*            spHostName;
(52)  static int              iErr;
(53)  static char             szErrBuf[256];
(54)  static int              iNumMessages = 10;
(55)  static char             szText[] = "This is a text message";
(56)
(57)  static void XATopicPub();
(58)  static void XATopicSub();
(59)
(60)  #define MSCLIENT_DLL_NAME "stc_msclient.dll"
(61)
(62)  /* Check for errors. */
(63)  static void check_error(int err, char* errBuf, int exitnow)
(64)  {
(65)       if (err){
(66)            printf("ERROR:0x%x - %s\n", err, errBuf);
(67)            if (exitnow)
(68)                 exit(1);
(69)       }
(70)  }
(71)
(72)
(73)  int main(int argc, char *argv[]) {
(74)       int           c;
(75)       char          cOption = 0;
(76)
(77)       spHostName = localHost;
(78)
(79)       while((c = getopt(argc, argv, ":p:h:P:H:cCUu")) != -1) {
(80)            switch(c){
(81)            case 'p':
(82)            case 'P':
(83)                 susPort = atoi(optarg);  /* setup the port number */
(84)                 break;
(85)            case 'h':
(86)            case 'H':
(87)                 spHostName = optarg;     /* setup the hostname */
(88)                 break;
(89)            case 'U':
(90)            case 'u':
(91)                 cOption = 'u';           /* run as a producer */
(92)                 break;
(93)            case 'c':
(94)            case 'C':
(95)                 cOption = 'c';           /* run as a consumer */
```

```
(96)                 break;
(97)            case ':':
(98)            case '?':
(99)                 printf("\nSYNOPSIS\n");
(100)                printf("%s %s\n", argv[0], optionProducer);
(101)                printf("%s %s\n", argv[0], optionConsumer);
(102)                printf("%s\n", optdescription);
(103)                exit(1);
(104)                break;
(105)           }
(106)     }
(107)
(108)     if (cOption == 'u'){
(109)          XATopicPub();          /* invoke producer */
(110)     } else if (cOption == 'c'){
(111)          XATopicSub();         /* invoke consumer */
(112)     } else {
(113)          printf("\nSYNOPSIS\n");
(114)          printf("%s %s\n", argv[0], optionProducer);
(115)          printf("%s %s\n", argv[0], optionConsumer);
(116)          printf("%s\n", optdescription);
(117)          exit(1);
(118)     }
(119) }
(120)
(121) /*
(122)  * ============================================================
(123)  * Publish Message
(124)  * This routine publishes iNumMessages to the topic
(125)  * ============================================================
(126) */
(127) static void PublishMessage(SBYN_TopicPublisher* pPublisher,
(128) SBYN_Message* pMessage, int iNumMessages)
(129) {
(130)     int ii;
(131)     for ( ii = 0; ii < iNumMessages; ii++){
(132)          SetIntProperty(pMessage, (char*)"Sequence", ii, &iErr,
(133)          szErrBuf);
(134)          check_error(iErr, szErrBuf, 1);
(135)          printf("Sending Message: Sequence number %d\n", ii);
(136)          TopicPublisherPublish(pPublisher, pMessage, &iErr, szErrBuf);
(137)          check_error(iErr, szErrBuf, 1);
(138)     }
(139) }
(140)
(141)
(142) /*
(143)  * ============================================================
(144)  * Receive Message
(145)  * This routine block on receiving message for maximum iWait
(146)  *   seconds before return.
(147)  * ============================================================
(148) */
(149) static int SubscriberReceiveMessage(SBYN_TopicSubscriber* pSub)
(150) {
(151)     int iMsgCount = 0;
(152)     SBYN_Message* pRMsg = 0;
(153)     char szUserInput[8];
(154)     printf("Waiting for message ... \n");
(155)     do {
(156)          pRMsg = TopicSubscriberReceive(pSub, &iErr, szErrBuf);
(157)          printf("Received Message %d\n", iMsgCount);
(158)          check_error(iErr, szErrBuf, 1);
(159)          iMsgCount++;
```

```
(160)           if (iMsgCount >= iNumMessages){
(161)                printf("Enter 'r' for receiving more message, 'q' for
(162)                exit\n");
(163)                scanf("%s", szUserInput);
(164)                iMsgCount = 0;
(165)           }
(166)
(167)      } while(szUserInput[0] != 'q' && szUserInput[0] != 'Q');
(168)      return iMsgCount;
(169) }
(170)
(171) /*
(172)  * ==========================================================
(173)  * Topic Publisher
(174)  * This routine demonstrates how to publish to a topic.
(175)  * ==========================================================
(176) */
(177) void XATopicPub()
(178) {
(179)      SBYN_XATopicConnectionFactory* pXATcf = 0;
(180)      SBYN_Connection*               pConnection = 0;
(181)      SBYN_Session*                  pXATopicSession = 0;
(182)      SBYN_Session*                  pTopicSession = 0;
(183)      SBYN_Destination*              pTopic = 0;
(184)      SBYN_XAResource*               pXATopicResource;
(185)      SBYN_XAResource*               pXATopicResourceTmp;
(186)      SBYN_TopicPublisher*           pTopicPublisher;
(187)      SBYN_Message*                  pMessage;
(188)      SBYN_Xid*                      pXid;
(189)      char                           pTopicName[] = "XAPubSubSample";
(190)
(191)      /* create XA connection factory */
(192)     pXATcf = CreateXATopicConnectionFactory(spHostName, susPort, &iErr,
(193)      szErrBuf);
(194)      check_error(iErr, szErrBuf, 1);
(195)
(196)      /* create XA connection */
(197)      pConnection = CreateXATopicConnection(pXATcf, &iErr, szErrBuf);
(198)      check_error(iErr, szErrBuf, 1);
(199)
(200)      /* set client ID */
(201)      ConnectionSetClientID(pConnection, (char*)"eGate{7E527692-770A-
(202)      11D5-B139-935EB6E85DBD}", &iErr, szErrBuf);
(203)      check_error(iErr, szErrBuf, 1);
(204)
(205)      /* create XA session */
(206)      pXATopicSession = XAConnectionCreateXATopicSession(pConnection,
(207)      &iErr, szErrBuf);
(208)      check_error(iErr, szErrBuf, 1);
(209)
(210)      /* get session */
(211)      pTopicSession = XASessionGetTopicSession(pXATopicSession, &iErr,
(212)      szErrBuf);
(213)      check_error(iErr, szErrBuf, 1);
(214)
(215)      /* get XA resource */
(216)      pXATopicResource = XASessionGetXAResource(pXATopicSession, &iErr,
(217)      szErrBuf);
(218)      check_error(iErr, szErrBuf, 1);
(219)
(220)      /* get XA resource */
(221)      pXATopicResourceTmp = XASessionGetXAResource(pXATopicSession,
(222)      &iErr, szErrBuf);
(223)      check_error(iErr, szErrBuf, 1);
```

```
(224)
(225)       /* create a Topic */
(226)       pTopic = SessionCreateTopic(pTopicSession, pTopicName, &iErr,
(227)       szErrBuf);
(228)       check_error(iErr, szErrBuf, 1);
(229)
(230)       /* create a publisher */
(231)       pTopicPublisher = SessionCreatePublisher(pTopicSession, pTopic,
(232)       &iErr, szErrBuf);
(233)       check_error(iErr, szErrBuf, 1);
(234)
(235)       /* connection start */
(236)       ConnectionStart(pConnection, &iErr, szErrBuf);
(237)       check_error(iErr, szErrBuf, 1);
(238)
(239)       /* create xa id */
(240)       pXid = XACreateXid((char*)MSCLIENT_DLL_NAME, &iErr, szErrBuf);
(241)       check_error(iErr, szErrBuf, 1);
(242)
(243)       /* associate the global transaction with the resource */
(244)       XAResourceStart(pXATopicResource, pXid, SBYN_TMNOFLAGS, &iErr,
(245)       szErrBuf);
(246)       check_error(iErr, szErrBuf, 1);
(247)
(248)       /* create a meesage */
(249)       pMessage = SessionCreateTextMessage(pXATopicSession, &iErr,
(250)       szErrBuf);
(251)       check_error(iErr, szErrBuf, 1);
(252)
(253)       /* set mode to r/w */
(254)       ClearBody(pMessage, &iErr, szErrBuf);
(255)       check_error(iErr, szErrBuf, 1);
(256)
(257)       /* write bytes */
(258)       SetText(pMessage, (char*)szText, &iErr, szErrBuf);
(259)       check_error(iErr, szErrBuf, 1);
(260)
(261)       /* publish message */
(262)       printf("Sending %d messages\n", iNumMessages);
(263)       PublishMessage(pTopicPublisher, pMessage, iNumMessages);
(264)
(265)       /* xaEnd */
(266)       XAResourceEnd(pXATopicResource, pXid, SBYN_TMSUCCESS, &iErr,
(267)       szErrBuf);
(268)       check_error(iErr, szErrBuf, 1);
(269)
(270)       /* ================================================
(271)        * Prepare-Rollback
(272)        * ================================================
(273)        */
(274)       /* xaPrepare */
(275)       XAResourcePrepare(pXATopicResource, pXid, &iErr, szErrBuf);
(276)       check_error(iErr, szErrBuf, 1);
(277)
(278)       /* xaRollBack */
(279)       printf("Rolling back %d message\n", iNumMessages);
(280)       XAResourceRollback(pXATopicResource, pXid, &iErr, szErrBuf);
(281)       check_error(iErr, szErrBuf, 1);
(282)
(283)
(284)       /* ================================================
(285)        * Prepare-Commit
(286)        * ================================================
(287)        */
```

```
(288)
(289)      /* xa start */
(290)      XAResourceStart(pXATopicResource, pXid, SBYN_TMNOFLAGS, &iErr,
(291)      szErrBuf);
(292)      check_error(iErr, szErrBuf, 1);
(293)
(294)      /* send message */
(295)      printf("Sending %d messages\n", iNumMessages);
(296)      PublishMessage(pTopicPublisher, pMessage, iNumMessages);
(297)
(298)      /* xaEnd */
(299)      XAResourceEnd(pXATopicResource, pXid, SBYN_TMSUCCESS, &iErr,
(300)      szErrBuf);
(301)      check_error(iErr, szErrBuf, 1);
(302)
(303)      /* xaPrepare */
(304)      if (SBYN_XA_OK != XAResourcePrepare(pXATopicResource, pXid, &iErr,
(305)      szErrBuf))
(306)      {
(307)           printf("ERROR: XAResourcePrepare failed\n");
(308)      }
(309)      check_error(iErr, szErrBuf, 1);
(310)
(311)      /* xa commit */
(312)      printf("Session Commit...\n");
(313)      XAResourceCommit(pXATopicResource, pXid, TRUE, &iErr, szErrBuf);
(314)      check_error(iErr, szErrBuf, 1);
(315)
(316)      /* Close and clean up. */
(317)      TopicPublisherClose(pTopicPublisher, &iErr, szErrBuf);
(318)      check_error(iErr, szErrBuf, 1);
(319)      SessionClose(pXATopicSession, &iErr, szErrBuf);
(320)      check_error(iErr, szErrBuf, 1);
(321)      ConnectionClose(pConnection, &iErr, szErrBuf);
(322)      check_error(iErr, szErrBuf, 1);
(323)      DeleteMessage(pMessage, &iErr, szErrBuf);
(324)      DeleteTopicPublisher(pTopicPublisher, &iErr, szErrBuf);
(325)  DeleteXAResource(pXATopicResource, &iErr, szErrBuf);
(326)      DeleteXid(pXid, &iErr, szErrBuf);
(327)      DeleteSession(pXATopicSession, &iErr, szErrBuf); /* delete session
(328)      & resource */
(329)      DeleteDestination(pTopic, &iErr, szErrBuf);
(330)      DeleteConnection(pConnection, &iErr, szErrBuf);
(331)      DeleteXATopicConnectionFactory(pXATcf, &iErr, szErrBuf);
(332) }
(333)
(334) /*
(335)  * ========================================================
(336)  * Topic Subscriber
(337)  * This routine demonstrates how to subsribe a message from a
(338)  *  topic.
(339)  * ========================================================
(340) */
(341) void XATopicSub()
(342) {
(343)      SBYN_XATopicConnectionFactory* pXATcf = 0;
(344)      SBYN_Connection*               pConnection = 0;
(345)      SBYN_Session*                  pXATopicSession = 0;
(346)      SBYN_Session*                  pTopicSession = 0;
(347)      SBYN_XAResource*               pTopicResource = 0;
(348)      SBYN_Destination*              pTopic = 0;
(349)      SBYN_Message*                  pReceivedMessage = 0;
(350)      SBYN_XAResource*               pXATopicResource = 0;
(351)      SBYN_TopicSubscriber*          pTopicSubscriber = 0;
```

```
(352)      SBYN_Message*                  pMessage = 0;
(353)      SBYN_Xid*                      pXid = 0;
(354)     char                          pTopicName[] = "eGateXAPubSubSample";
(355)      int                            iNumReceived = 0;
(356)      char                           szUserInput[8];
(357)      /* create XA connection factory */
(358)     pXATcf = CreateXATopicConnectionFactory(spHostName, susPort, &iErr,
(359)      szErrBuf);
(360)      check_error(iErr, szErrBuf, 1);
(361)
(362)      /* create XA connection */
(363)      pConnection = CreateXATopicConnection(pXATcf, &iErr, szErrBuf);
(364)      check_error(iErr, szErrBuf, 1);
(365)
(366)      /* set client ID */
(367)      ConnectionSetClientID(pConnection, (char*)"eGate{7E527692-770A-
(368)      11D5-B139-3456789}", &iErr, szErrBuf);
(369)      check_error(iErr, szErrBuf, 1);
(370)
(371)      /* create XA session */
(372)      pXATopicSession = XAConnectionCreateXATopicSession(pConnection,
(373)      &iErr, szErrBuf);
(374)      check_error(iErr, szErrBuf, 1);
(375)
(376)      /* get session */
(377)      pTopicSession = XASessionGetTopicSession(pXATopicSession, &iErr,
(378)      szErrBuf);
(379)      check_error(iErr, szErrBuf, 1);
(380)
(381)      /* get XA resource */
(382)      pXATopicResource = XASessionGetXAResource(pXATopicSession, &iErr,
(383)      szErrBuf);
(384)      check_error(iErr, szErrBuf, 1);
(385)
(386)      /* create a Topic */
(387)      pTopic = SessionCreateTopic(pTopicSession, pTopicName, &iErr,
(388)      szErrBuf);
(389)      check_error(iErr, szErrBuf, 1);
(390)
(391)      /* create a subscriber */
(392)      //pTopicSubscriber = SessionCreateDurableSubscriber(pTopicSession,
(393)      pTopic, (char*)"XATopicSubscriber", &iErr, szErrBuf);
(394)      pTopicSubscriber = SessionCreateSubscriber(pTopicSession, pTopic,
(395)      &iErr, szErrBuf);
(396)      check_error(iErr, szErrBuf, 1);
(397)
(398)      /* connection start */
(399)      ConnectionStart(pConnection, &iErr, szErrBuf);
(400)      check_error(iErr, szErrBuf, 1);
(401)
(402)      /* start xa resource */
(403)      pXid = XACreateXid((char*)MSCLIENT_DLL_NAME, &iErr, szErrBuf);
(404)      check_error(iErr, szErrBuf, 1);
(405)
(406)
(407)
(408)      /* Receive all the messages on the topic and return the number of
(409)      messages received */
(410)      //iNumReceived = SubscriberReceiveMessage(pTopicSubscriber);
(411)      printf("Receiving messages...\n");
(412)      do {
(413)   int iMsgCount = 0;
(414)           SBYN_Message* pRMsg = 0;
(415)           /* associate the global transaction with the resource */
```

```
(416)   XAResourceStart(pXATopicResource, pXid, SBYN_TMNOFLAGS, &iErr,
(417)   szErrBuf);
(418)   check_error(iErr, szErrBuf, 1);
(419)           while(iMsgCount < iNumMessages){
(420)                pRMsg = TopicSubscriberReceive(pTopicSubscriber, &iErr,
(421)                szErrBuf);
(422)                printf("Received Message %d\n", iMsgCount);
(423)                iMsgCount++;
(424)           }
(425)
(426)           /* xaEnd */
(427)   XAResourceEnd(pXATopicResource, pXid, SBYN_TMSUCCESS, &iErr,
(428)   szErrBuf);
(429)   check_error(iErr, szErrBuf, 1);
(430)           XAResourceCommit(pXATopicResource, pXid, TRUE, &iErr,
(431)   szErrBuf);
(432)           printf("Enter 'r' for receiving more message, 'q' for
(433)   exit\n");
(434)           scanf("%s", szUserInput);
(435)           iMsgCount = 0;
(436)      } while(szUserInput[0] != 'q' && szUserInput[0] != 'Q');
(437)
(438)      /* Session commit */
(439)      SessionCommit(pTopicSession,  &iErr, szErrBuf);
(440)
(441)      /* close and delete objects */
(442)      TopicSubscriberClose(pTopicSubscriber, &iErr, szErrBuf);
(443)      SessionUnsubscribe(pXATopicSession, (char*)"TopicSubscriber",
(444)      &iErr, szErrBuf);
(445)      SessionClose(pXATopicSession, &iErr, szErrBuf);
(446)      ConnectionClose(pConnection, &iErr, szErrBuf);
(447)
(448)      /* delete objects */
(449)      DeleteDestination(pTopic, &iErr, szErrBuf);
(450)      DeleteTopicSubscriber(pTopicSubscriber, &iErr, szErrBuf);
(451)  DeleteXAResource(pXATopicResource, &iErr, szErrBuf);
(452)      DeleteXid(pXid, &iErr, szErrBuf);
(453)      DeleteSession(pXATopicSession, &iErr, szErrBuf); /* delete XA
(454)      resource and session */
(455)      DeleteConnection(pConnection, &iErr, szErrBuf);
(456)      DeleteXATopicConnectionFactory(pXATcf, &iErr, szErrBuf);

       }
```

Chapter 8

Client Libraries for the C API

This chapter provides a detailed discussion of the Application Program Interfaces (APIs) for SeeBeyond JMS.

For C

- **The C API** on page 173
- **Error Codes and Messages in the C API** on page 273

Note: *Double-byte character sets (DBCS) are not supported for the C API on Solaris and HP-UX operating systems.*

The SeeBeyond Message Service includes additional API support in the following languages:

For Java

- **The JMS API for the SeeBeyond Message Service** on page 142

For COM+

- **The COM+ API for the SeeBeyond Message Service** on page 76

For C++

The C++ API for the SeeBeyond Message Service on page 299

The Difference Between the Java API and the C API

Differences Between JMS Java API and SeeBeyond JMS C API on page 274

8.1 The C API

The **C** API for SeeBeyond JMS, in the library **stc_msclient.dll**, is a wrapper around the **C++** API for SeeBeyond JMS. This section provides the following information:

- **Architectural Overview** on page 174
- **Structures** on page 175
- **Interfaces** on page 178
- Detailed information on the following:
  - Function prototypes for each interface
  - **Destructor Methods** on page 264
  - **The WString Helper Interface** on page 270
  - **The WStringList Helper Interface** on page 272
- **Error Codes and Messages in the C API** on page 273.

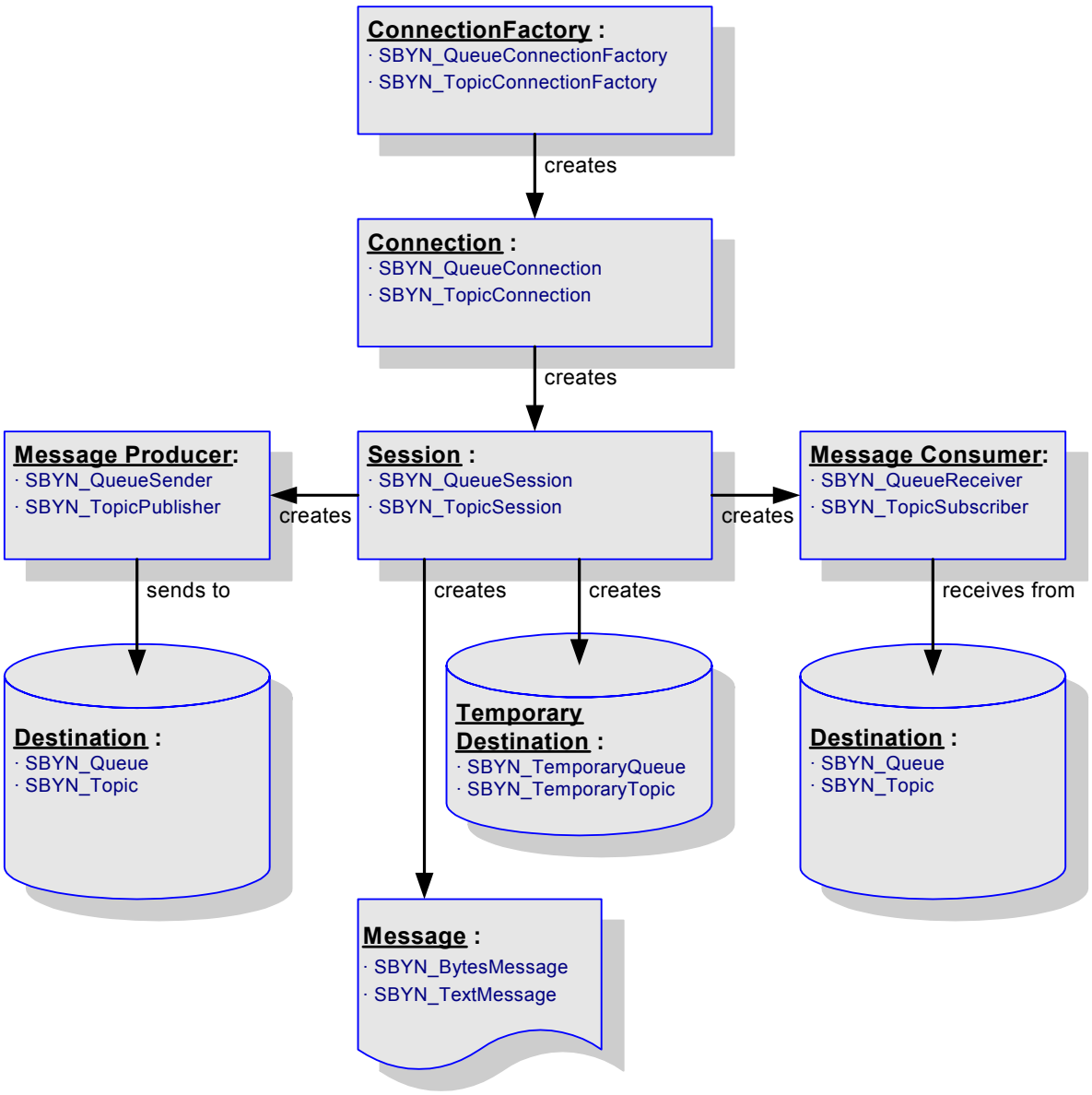
Also see:

- **"Sample Code for Using JMS in C" on page 144**.

8.1.1. Architectural Overview

The standard JMS API object model is followed in **C**, as illustrated in Figure 26.

Figure 26 JMS **C** Object Model



8.1.2. Structures

The **C** API for SeeBeyond JMS comprises the following structures:

Table 8 Structures in **C** API for SeeBeyond JMS

For Point-to-Point (P2P) Messaging	For Publish/Subscribe (Pub/Sub) Messaging
struct SBYN_QueueConnectionFactory	struct SBYN_TopicConnectionFactory
struct SBYN_QueueConnection	struct SBYN_TopicConnection
struct SBYN_QueueSession	struct SBYN_TopicSession
struct SBYN_QueueSender	struct SBYN_TopicPublisher
struct SBYN_QueueReceiver	struct SBYN_TopicSubscriber
struct SBYN_Queue	struct SBYN_Topic
struct SBYN_TemporaryQueue	struct SBYN_TemporaryTopic
struct SBYN_QueueRequestor	struct SBYN_TopicRequestor
For P2P and Pub/Sub Messaging	
struct SBYN_Destination	
struct SBYN_Message	
struct SBYN_BytesMessage	
struct SBYN_TextMessage	
struct SBYN_ConnectionMetaData	
struct SBYN_WString	
struct SBYN_WStringList	

8.1.3. Constants

The **C** API for SeeBeyond JMS defines values for the following types of constants:

- **DeliveryMode Constants** on page 176
- **DestinationType Constants** on page 176
- **MessageType Constants** on page 176
- **Session Constants** on page 176
- **Transacted Constants** on page 177
- **Miscellaneous Constants Setting Message Class Defaults** on page 178
- **Other Miscellaneous Constants** on page 178

DeliveryMode Constants

Table 9 Values for DeliveryMode Constants

Name	
SBYN_NON_PERSISTENT	0
SBYN_PERSISTENT	1

NON_PERSISTENT

0 signifies non-persistent delivery mode. This mode maximizes performance, and should be used if an occasional lost message is tolerable.

PERSISTENT

1 signifies persistent delivery mode. This mode maximizes reliability, and should be used if the application will have problems if the message is lost in transit.

DestinationType Constants

Table 10 Values for DestinationType Constants

Name	
SBYN_DESTINATION_TYPE_QUEUE	0
SBYN_DESTINATION_TYPE_TEMPORARYQUEUE	1
SBYN_DESTINATION_TYPE_TOPIC	2
SBYN_DESTINATION_TYPE_TEMPORARYTOPIC	3

MessageType Constants

Table 11 Values for MessageType Constants

Name	
SBYN_MESSAGE_TYPE_MESSAGE	0
SBYN_MESSAGE_TYPE_TEXT	1
SBYN_MESSAGE_TYPE_BYTES	2

Session Constants

Table 12 Values for Session Constants

Name	
SBYN_AUTO_ACKNOWLEDGE	1
SBYN_CLIENT_ACKNOWLEDGE	2

Table 12 Values for Session Constants

Name	
SBYN_DUPS_OK_ACKNOWLEDGE	3

AutoAcknowledge Mode

1 signifies auto-acknowledgment: The session automatically acknowledges a client's receipt of a message either upon its successful return from a call to **receive** or upon successful return of the MessageListener it has called to process the message.

ClientAcknowledge Mode

2 signifies acknowledgment by client: A client acknowledges a message by calling the message's **acknowledge** method. Acknowledging a consumed message automatically acknowledges the receipt of all messages that have been delivered by its session.

DupsOKAcknowledge Mode

3 indicates that duplicates are acceptable, and instructs the session to lazily acknowledge message delivery This setting is likely to cause delivery of some duplicate messages if JMS fails, and should only be used by consumers that are tolerant of duplicate messages. The benefit is the reduction of session overhead, achieved by minimizing the work done to prevent duplicates.

Transacted Constants

Table 13 Values for Transacted Constants

Name	
SBYN_NON_TRANSACTED	0
SBYN_TRANSACTED	1

If a session is specified as being *transacted*, it supports a single series of transactions. A set of messages-received is grouped into an atomic unit of input, and a set of staged messages-to-be-sent is grouped into an atomic unit of output. When a transaction performs a *commit*, the atomic unit of input is acknowledged and the associated atomic unit of output is sent. If, instead, the transaction performs a *rollback*, all messages in the atomic unit of output are destroyed and the session's input is automatically recovered.

A transaction is completed only by a commit or by a rollback. The completion of a session's current transaction automatically begins the next transaction. In this way, a transacted session always has a current transaction within which work is done.

Miscellaneous Constants Setting Message Class Defaults

Table 14 Values for Miscellaneous Constants Setting Message Class Defaults

Name	
SBYN_DEFAULT_DELIVERY_MODE	1
SBYN_DEFAULT_PRIORITY	4
SBYN_DEFAULT_TIME_TO_LIVE	0

Default Setting for DeliveryMode

See **"DeliveryMode Constants" on page 176**.

Default Setting for Priority

JMS defines a ten-level priority value: **0** is lowest priority (least expedited) and **9** is highest. Clients should consider priorities **0** through **4** as gradations of normal priority and priorities **5** through **9** as gradations of expedited priority.

Default Setting for TimeToLive

Length of time that a produced message should be retained by the message system. Measured in milliseconds elapsed since its dispatch time. The default, **0**, has the special meaning of "retain forever"—that is, the message never expires on its own.

Other Miscellaneous Constants

Table 15 Values for Other Miscellaneous Constants

Name	Value
DEFAULT_PORT	18008
DEFAULT_SERVER_NAME	"localhost"
MSCLIENT_DLL_NAME	"stc_msclient.dll"
FALSE	0
TRUE	1

8.1.4. Interfaces

The following interfaces have defined prototypes:

8.1.5. The Message Interface

The **Message** interface defines methods for working with a **Message** object. The interface includes the following methods:

Acknowledge

Syntax

```
Acknowledge(pMsg, iError, pczError)
```

Description

Acknowledges the receipt of current and previous messages.

Parameters

Name	Type	Description
pMsg	SBYN_Message*	Pointer to the message.

Name	Type	Description
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

None.

ClearBody

Syntax

```
ClearBody(pMsg, iError, pczError)
```

Description

Clears the body of a message, leaving the message header values and property entries intact.

Parameters

Name	Type	Description
pMsg	SBYN_Message*	Pointer to the message.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

None.

ClearProperties

Syntax

```
ClearProperties(pMsg, iError, pczError)
```

Description

Clears the properties from a message, leaving the message header fields and body intact.

Parameters

Name	Type	Description
pMsg	SBYN_Message*	Pointer to the message.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

None.

PropertyExists

Syntax

```
PropertyExists(pMsg, pczName, iError, pczError)
```

Description

Checks whether a value for a specific property exists.

Parameters

Name	Type	Description
pMsg	SBYN_Message*	Pointer to the message.
pczName	char*	Pointer to the name of the property to check.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

bool
Returns **true** if the property value is defined; otherwise, returns **false**.

GetBooleanProperty

Syntax

```
GetBooleanProperty(pMsg, pczName, iError, pczError)
```

Description

Reads the value of the specified Boolean property in the specified message.

Parameters

Name	Type	Description
pMsg	SBYN_Message*	Pointer to the message.
pczName	char*	Pointer to the name of the property to check.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

SBYN_BOOL
The value of the property.

GetByteProperty

Syntax

```
GetByteProperty(pMsg, pczName, iError, pczError)
```

Description

Reads the value of the specified byte property in the specified message.

Parameters

Name	Type	Description
pMsg	SBYN_Message*	Pointer to the message.
pczName	char*	Pointer to the name of the property to check.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

unsigned char
The value of the property.

GetDoubleProperty

Syntax

```
GetDoubleProperty(pMsg, pczName, iError, pczError)
```

Description

Reads the value of the specified double (numeric) property in the specified message.

Parameters

Name	Type	Description
pMsg	SBYN_Message*	Pointer to the message.
pczName	char*	Pointer to the name of the property to check.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

double
The value of the property.

GetFloatProperty

Syntax

```
GetFloatProperty(pMsg, pczName, iError, pczError)
```

Description

Reads the value of the specified floating-point (numeric) property in the specified message.

Parameters

Name	Type	Description
pMsg	SBYN_Message*	Pointer to the message.
pczName	char*	Pointer to the name of the property to check.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

float
The value of the property.

GetIntProperty

Syntax

```
GetIntProperty(pMsg, pczName, iError, pczError)
```

Description

Reads the value of the specified integer (numeric) property in the specified message.

Parameters

Name	Type	Description
pMsg	SBYN_Message*	Pointer to the message.
pczName	char*	Pointer to the name of the property to check.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

int
The value of the property.

GetLongProperty

Syntax

```
GetLongProperty(pMsg, pczName, iError, pczError)
```

Description

Reads the value of the specified long (numeric) property in the specified message.

Parameters

Name	Type	Description
pMsg	SBYN_Message*	Pointer to the message.
pczName	char*	Pointer to the name of the property to check.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

long
The value of the property.

GetShortProperty

Syntax

```
GetShortProperty(pMsg, pczName, iError, pczError)
```

Description

Reads the value of the specified short (numeric) property in the specified message.

Parameters

Name	Type	Description
pMsg	SBYN_Message*	Pointer to the message.
pczName	char*	Pointer to the name of the property to check.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

short
The value of the property.

GetStringProperty

Syntax

```
GetStringProperty(pMsg, pczName, iError, pczError)
```

Description

Reads the value of the specified text property in the specified message.

Parameters

Name	Type	Description
pMsg	SBYN_Message*	Pointer to the message.
pczName	char*	Pointer to the name of the property to check.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

WString*

Pointer to a **WString** (wide string) object containing the value of the property.

SetBooleanProperty

Syntax

```
SetBooleanProperty(pMsg, pczName, bValue, iError, pczError)
```

Description

Writes a value for the specified Boolean property in the specified message.

Parameters

Name	Type	Description
pMsg	SBYN_Message*	Pointer to the message.
pczName	char*	Pointer to the property name.
bValue	bool	The value to be written.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

None.

SetByteProperty

Syntax

```
SetByteProperty(pMsg, pczName, cValue, iError, pczError)
```

Description

Writes a value for the specified byte property in the specified message.

Parameters

Name	Type	Description
pMsg	SBYN_Message*	Pointer to the message.
pczName	char*	Pointer to the property name.
cValue	unsigned char	The value to be written.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

None.

SetDoubleProperty

Syntax

```
SetDoubleProperty(pMsg, pczName, dblValue, iError, pczError)
```

Description

Writes a value for the specified double (numeric) property in the specified message.

Parameters

Name	Type	Description
pMsg	SBYN_Message*	Pointer to the message.
pczName	char*	Pointer to the property name.
dblValue	double	The value to be written.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

None.

SetFloatProperty

Syntax

```
SetFloatProperty(pMsg, pczName, fltValue, iError, pczError)
```

Description

Writes a value for the specified floating-point (numeric) property in the specified message.

Parameters

Name	Type	Description
pMsg	SBYN_Message*	Pointer to the message.
pczName	char*	Pointer to the property name.
fltValue	float	The value to be written.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

None.

SetIntProperty

Syntax

```
SetIntProperty(pMsg, pczName, iValue, iError, pczError)
```

Description

Writes a value for the specified integer (numeric) property in the specified message.

Parameters

Name	Type	Description
pMsg	SBYN_Message*	Pointer to the message.
pczName	char*	Pointer to the property name.
iValue	int	The value to be written.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

None.

SetLongProperty

Syntax

```
SetLongProperty(pMsg, pczName, lValue, iError, pczError)
```

Description

Writes a value for the specified long (numeric) property in the specified message.

Parameters

Name	Type	Description
pMsg	SBYN_Message*	Pointer to the message.
pczName	char*	Pointer to the property name.
lValue	long	The value to be written.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

None.

SetShortProperty

Syntax

```
SetShortProperty(pMsg, pczName, nValue, iError, pczError)
```

Description

Writes a value for the specified short (numeric) property in the specified message.

Parameters

Name	Type	Description
pMsg	SBYN_Message*	Pointer to the message.
pczName	char*	Pointer to the property name.
nValue	short	The value to be written.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

None.

SetStringProperty

Syntax

```
SetStringProperty(pMsg, pczName, pczValue, iError, pczError)
```

Description

Writes a value for the specified text property in the specified message.

Parameters

Name	Type	Description
pMsg	SBYN_Message*	Pointer to the message.
pczName	char*	Pointer to the property name.
pczValue	char*	The value to be written.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

None.

GetJMSCorrelationID

Syntax

```
GetJMSCorrelationID(pMsg, iError, pczError)
```

Description

Retrieves the correlation ID for the specified message.

Parameters

Name	Type	Description
pMsg	SBYN_Message*	Pointer to the message.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

WString*
Pointer to a **WString** (wide string) object containing the text.

GetJMSCorrelationIDAsBytes

Syntax

```
GetJMSCorrelationIDAsBytes(pMsg, iError, pczError)
```

Description

Retrieves the correlation ID for the specified message as an array of characters.

Parameters

Name	Type	Description
pMsg	SBYN_Message*	Pointer to the message.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

char*

Pointer to an array of characters containing the text.

GetJMSDeliveryMode

Syntax

```
GetJMSDeliveryMode(pMsg, iError, pczError)
```

Description

Retrieves the value of the **DeliveryMode** property for the specified message.

Parameters

Name	Type	Description
pMsg	SBYN_Message*	Pointer to the message.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

int

See **"DeliveryMode Constants" on page 176**.

GetJMSExpiration

Syntax

```
GetJMSExpiration(pMsg, iError, pczError)
```

Description

Retrieves the value of the timestamp set for the expiration of the specified message.

Parameters

Name	Type	Description
pMsg	SBYN_Message*	Pointer to the message.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

long
Timestamp.

GetJMSMessageID

Syntax

```
GetJMSMessageID(pMsg, iError, pczError)
```

Description

Retrieves the message ID of the specified message.

Parameters

Name	Type	Description
pMsg	SBYN_Message*	Pointer to the message.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

WString*
Pointer to a **WString** (wide string) object containing the text.

GetJMSPriority

Syntax

```
GetJMSPriority(pMsg, iError, pczError)
```

Description

Retrieves the priority level for the specified message.

Parameters

Name	Type	Description
pMsg	SBYN_Message*	Pointer to the message.
iError	int*	Pointer to the error number.

Name	Type	Description
pczError	char*	Pointer to the error message text.

Return Value

int
The priority level.

Also see **"Miscellaneous Constants Setting Message Class Defaults" on page 178**.

GetJMSRedelivered

Syntax

```
GetJMSRedelivered(pMsg, iError, pczError)
```

Description

Retrieves an indication of whether the specified message is being redelivered.

Parameters

Name	Type	Description
pMsg	SBYN_Message*	Pointer to the message.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

SBYN_BOOL
Returns **true** if the message is being redelivered; otherwise, returns **false**.

GetJMSReplyTo

Syntax

```
GetJMSReplyTo(pMsg, iError, pczError)
```

Description

Retrieves the **Destination** object where a reply to the specified message should be sent (for request/reply messaging).

Parameters

Name	Type	Description
pMsg	SBYN_Message*	Pointer to the message.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

SBYN_Destination*
Pointer to the **Destination** object.

GetJMSTimestamp

Syntax

```
GetJMSTimestamp(pMsg, iError, pczError)
```

Description

Retrieves the timestamp of the specified message.

Parameters

Name	Type	Description
pMsg	SBYN_Message*	Pointer to the message.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

long
Timestamp.

GetJMSType

Syntax

```
GetJMSType(pMsg, iError, pczError)
```

Description

Gets the message type identifier supplied by the client when the message was sent.

Parameters

Name	Type	Description
pMsg	SBYN_Message*	Pointer to the message.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

WString*
Pointer to a **WString** (wide string) object containing the text.

SetJMSCorrelationID

Syntax

```
SetJMSCorrelationID(pMsg, pczValue, iError, pczError)
```

Description

Sets the correlation ID for the specified message.

Parameters

Name	Type	Description
pMsg	SBYN_Message*	Pointer to the message.
pczValue	char*	Pointer to the text string containing the correlation ID.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

None.

SetJMSCorrelationIDAsBytes

Syntax

```
SetJMSCorrelationIDAsBytes(pMsg, pczValue, iError, pczError)
```

Description

Sets the correlation ID for the specified message.

Parameters

Name	Type	Description
pMsg	SBYN_Message*	Pointer to the message.
pczValue	char*	Pointer to the character array containing the bytes for the correlation ID.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

None.

SetJMSDeliveryMode

Syntax

```
SetJMSDeliveryMode(pMsg, value, iValue, iError, pczError)
```

Description

Sets the delivery mode for the specified message. See **"DeliveryMode Constants" on page 176**.

Parameters

Name	Type	Description
pMsg	SBYN_Message*	Pointer to the message.
iValue	int	Value corresponding to the delivery mode.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

None.

SetJMSExpiration

Syntax

```
SetJMSExpiration(pMsg, lValue, iError, pczError)
```

Description

Sets the timestamp at which the specified message is due to expire.

Parameters

Name	Type	Description
pMsg	SBYN_Message*	Pointer to the message.
lValue	long	Timestamp of the expiration.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

None.

SetJMSMessageID

Syntax

```
SetJMSMessageID(pMsg, pczValue, iError, pczError)
```

Description

Sets the message ID of the specified message.

Parameters

Name	Type	Description
pMsg	SBYN_Message*	Pointer to the message.
pczValue	char*	Pointer to the text string containing the message ID.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

None.

SetJMSPriority

Syntax

```
SetJMSPriority(pMsg, iValue, iError, pczError)
```

Description

Sets the priority level for the specified message.

Parameters

Name	Type	Description
pMsg	SBYN_Message*	Pointer to the message.
iValue	int	The priority level.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

None.

SetJMSRedelivered

Syntax

```
SetJMSRedelivered(pMsg, fValue, iError, pczError)
```

Description

Determines whether to flag the specified message as being redelivered. Used, for example, to specify redelivery for a message that has been sent but not acknowledged.

Parameters

Name	Type	Description
pMsg	SBYN_Message*	Pointer to the message.
fValue	SBYN_BOOL	Flag: If **true**, the message is being redelivered.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

None.

SetJMSReplyTo

Syntax

```
SetJMSReplyTo(pMsg, pDest, iError, pczError)
```

Description

Sets the **Destination** object where a reply to the specified message should be sent.

Parameters

Name	Type	Description
pMsg	SBYN_Message*	Pointer to the message.
pDest	SBYN_Destination*	Pointer to the **Destination** object.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

None.

SetJMSTimestamp

Syntax

```
SetJMSTimestamp(pMsg, lValue, iError, pczError)
```

Description

Sets the timestamp (JMSTimestamp header field) that the specified message was handed off to a provider to be sent. Note that this is not necessarily the time the message is actually transmitted; the actual **send** can occur later because of transactions or other client-side queueing of messages.

Parameters

Name	Type	Description
pMsg	SBYN_Message*	Pointer to the message.
lValue	long	Timestamp to be set.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

None.

SetJMSType

Syntax

```
SetJMSType(pMsg, pczJMSType, iError, pczError)
```

Description

Sets the JMSType header field, which is often used to identify the message structure and the payload type.

Parameters

Name	Type	Description
pMsg	SBYN_Message*	Pointer to the message.
pczJMSType	char*	Pointer to the text string containing the data.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

None.

8.1.6. The Extended Message Interface

The extended message interface includes the following function:

- **GetMessageType** on page 199

GetMessageType

Syntax

```
GetMessageType(pMsg)
```

Description

Retrieves the message type (bytesmessage, textmessage, and so forth).

Parameters

Name	Type	Description
pMsg	SBYN_Message*	Pointer to the message.

Return Value

MessageType_t
See **"MessageType Constants" on page 176**.

8.1.7. BytesMessage Methods

A **BytesMessage** object is used for messages containing a stream of uninterpreted bytes. The receiver of the message supplies the interpretation of the bytes.

The **BytesMessage** methods include the following:

- **WriteUTF** on page 211

ReadBoolean

Syntax

```
ReadBoolean(pMsg, iError, pczError)
```

Description

Reads a Boolean value from the **BytesMessage** stream.

Parameters

Name	Type	Description
pMsg	SBYN_Message*	Pointer to the message.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

SBYN_BOOL
The value read from the **BytesMessage** stream.

ReadByte

Syntax

```
ReadByte(pMsg, iError, pczError)
```

Description

Reads a single unsigned character from the **BytesMessage** stream.

Parameters

Name	Type	Description
pMsg	SBYN_Message*	Pointer to the message.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

unsigned char
The value read from the **BytesMessage** stream.

ReadBytes

Syntax

```
ReadBytes(pMsg, pczValue, iLength, iError, pczError)
```

Description

Reads a portion of the **BytesMessage** stream into a buffer. If the length of array value is less than the bytes remaining to be read from the stream, the array is filled.
A subsequent call reads the next increment, and so on.

If there are fewer bytes remaining in the stream than the length of array value, the bytes are read into the array, and the return value (total number of bytes read) is less than the length of the array, indicating that there are no more bytes left to be read from the stream. The next read of the stream returns **-1**.

Parameters

Name	Type	Description
pMsg	SBYN_Message*	Pointer to the message.
pczValue	unsigned char*	Pointer to the buffer into which the bytes are read.
iLength	int	The number of bytes to read; must be less than the length of the buffer.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

int
Number of bytes read into the buffer; or **-1** if all bytes were read previously.

ReadChar

Syntax

```
ReadChar(pMsg, iError, pczError)
```

Description

Reads a single Unicode character from the **BytesMessage** stream.

Parameters

Name	Type	Description
pMsg	SBYN_Message*	Pointer to the message.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

unsigned short
The value read from the **BytesMessage** stream.

ReadDouble

Syntax

```
ReadDouble(pMsg, iError, pczError)
```

Description

Reads a double numeric value from the **BytesMessage** stream.

Parameters

Name	Type	Description
pMsg	SBYN_Message*	Pointer to the message.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

double
The value read from the **BytesMessage** stream.

ReadFloat

Syntax

```
ReadFloat(pMsg, iError, pczError)
```

Description

Reads a floating-point numeric value from the **BytesMessage** stream.

Parameters

Name	Type	Description
pMsg	SBYN_Message*	Pointer to the message.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

float
The value read from the **BytesMessage** stream.

ReadInt

Syntax

```
ReadInt(pMsg, iError, pczError)
```

Description

Reads a signed integer value from the **BytesMessage** stream.

Parameters

Name	Type	Description
pMsg	SBYN_Message*	Pointer to the message.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

int
The value read from the **BytesMessage** stream.

ReadLong

Syntax

```
ReadLong(pMsg, iError, pczError)
```

Description

Reads a signed long integer from the **BytesMessage** stream.

Parameters

Name	Type	Description
pMsg	SBYN_Message*	Pointer to the message.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

long
The value read from the **BytesMessage** stream.

ReadShort

Syntax

```
ReadShort(pMsg, iError, pczError)
```

Description

Reads a signed short integer from the **BytesMessage** stream.

Parameters

Name	Type	Description
pMsg	SBYN_Message*	Pointer to the message.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

short
The value read from the **BytesMessage** stream.

ReadUnsignedByte

Syntax

```
ReadUnsignedByte(pMsg, iError, pczError)
```

Description

Reads an unsigned short integer from the **BytesMessage** stream.

Parameters

Name	Type	Description
pMsg	SBYN_Message*	Pointer to the message.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

int
The value read from the **BytesMessage** stream.

ReadUnsignedShort

Syntax

```
ReadUnsignedShort(pMsg, iError, pczError)
```

Description

Reads an unsigned short integer from the **BytesMessage** stream.

Parameters

Name	Type	Description
pMsg	SBYN_Message*	Pointer to the message.
iError	int*	Pointer to the error number.

Name	Type	Description
pczError	char*	Pointer to the error message text.

Return Value

int
The value read from the **BytesMessage** stream.

ReadUTF

Syntax

```
ReadUTF(pMsg, iError, pczError)
```

Description

Reads the value of a string that has been encoded using a modified UTF-8 format from the **BytesMessage** stream.

Parameters

Name	Type	Description
pMsg	SBYN_Message*	Pointer to the message.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

WString*
Pointer to a **WString** (wide string) object containing the text from the **BytesMessage** stream.

Reset

Syntax

```
Reset(pMsg, iError, pczError)
```

Description

Puts the message body into read-only mode and repositions the stream of bytes to the beginning.

Parameters

Name	Type	Description
pMsg	SBYN_Message*	Pointer to the message.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

None.

WriteBoolean

Syntax

```
WriteBoolean(pMsg, fValue, iError, pczError)
```

Description

Writes a Boolean value to the **BytesMessage** stream.

Parameters

Name	Type	Description
pMsg	SBYN_Message*	Pointer to the message.
fValue	SBYN_BOOL	The value to be written.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

None.

WriteByte

Syntax

```
WriteByte(pMsg, cValue, iError, pczError)
```

Description

Writes a single byte (unsigned char) to the **BytesMessage** stream.

Parameters

Name	Type	Description
pMsg	SBYN_Message*	Pointer to the message.
cValue	unsigned char	The value to be written.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

None.

WriteBytes

Syntax

```
WriteBytes(pMsg, pczValue, iError, pczError)
```

Description

Writes an array of bytes (unsigned char values) to the **BytesMessage** stream.

Parameters

Name	Type	Description
pMsg	SBYN_Message*	Pointer to the message.
pczValue	unsigned char*	Pointer to the value to be written.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

None.

WriteBytesEx

Syntax

```
WriteBytesEx(pMsg, pczValue, iOffset, iLength, iError, pczError)
```

Description

Writes a portion of a byte array (unsigned char values) to the **BytesMessage** stream. For example, to extract “nag” from “manager”, set iOffset=2 and iLength=3.

Parameters

Name	Type	Description
pMsg	SBYN_Message*	Pointer to the message.
pczValue	unsigned char*	Pointer to the value to be written.
iOffset	int	The initial offset within the byte array.
iLength	int	The number of bytes to use.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

None.

WriteChar

Syntax

```
WriteChar(pMsg, cValue, iError, pczError)
```

Description

Writes an unsigned short integer to the **BytesMessage** stream.

Parameters

Name	Type	Description
pMsg	SBYN_Message*	Pointer to the message.
cValue	unsigned short	The value to be written.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

None.

WriteDouble

Syntax

```
WriteDouble(pMsg, dblValue, iError, pczError)
```

Description

Writes a double numeric value to the **BytesMessage** stream.

Parameters

Name	Type	Description
pMsg	SBYN_Message*	Pointer to the message.
dblValue	double	The value to be written.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

None.

WriteFloat

Syntax

```
WriteFloat(pMsg, fltValue, iError, pczError)
```

Description

Writes a floating-point numeric value to the **BytesMessage** stream.

Parameters

Name	Type	Description
pMsg	SBYN_Message*	Pointer to the message.
fltValue	float	The value to be written.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

None.

WriteInt

Syntax

```
WriteInt(pMsg, iValue, iError, pczError)
```

Description

Writes an integer numeric value to the **BytesMessage** stream.

Parameters

Name	Type	Description
pMsg	SBYN_Message*	Pointer to the message.
iValue	int	The value to be written.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

None.

WriteLong

Syntax

```
WriteLong(pMsg, lValue, iError, pczError)
```

Description

Writes a long numeric value to the **BytesMessage** stream.

Parameters

Name	Type	Description
pMsg	SBYN_Message*	Pointer to the message.
lValue	int	The value to be written.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

None.

WriteShort

Syntax

```
WriteShort(pMsg, nValue, iError, pczError)
```

Description

Writes a short numeric value to the **BytesMessage** stream.

Parameters

Name	Type	Description
pMsg	SBYN_Message*	Pointer to the message.
nValue	short	The value to be written.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

None.

WriteUTF

Syntax

```
WriteUTF(pMsg, pczValue, iError, pczError)
```

Description

Writes a character string to the BytesMessage stream using UTF-8 encoding.

Parameters

Name	Type	Description
pMsg	SBYN_Message*	Pointer to the message.
pczValue	char*	Pointer to the value to be written.

Name	Type	Description
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

None.

8.1.8. TextMessage Methods

A TextMessage is used to send a message containing text. It adds a text message body. When a client receives a TextMessage, it is in read-only mode. If an attempt is made to write to the message while in read-only mode, an error is returned. However, if ClearBody is called first, then message can be then read from and written to.

The TextMessage functions include the following:

- **GetText** on page 212
- **SetText** on page 212

GetText

Syntax

```
GetText(pMsg, iError, pczError)
```

Description

Retrieves the string containing the data associated with the message.

Parameters

Name	Type	Description
pMsg	SBYN_Message*	Pointer to the message.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

char*
Pointer to the string. The default value is **0** (null).

SetText

Syntax

```
SetText(pMsg, pczBuffer, iError, pczError)
```

Description

Sets the string containing the data associated with the message.

Parameters

Name	Type	Description
pMsg	SBYN_Message*	Pointer to the message.
pczBuffer	char*	Pointer to the text string.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

None.

8.1.9. The QueueConnectionFactory Interface

Using point-to-point messaging, a client uses a **QueueConnectionFactory** object to create **QueueConnection** objects.

The **QueueConnectionFactory** interface includes the following methods:

- **CreateQueueConnectionFactory** on page 213
- **CreateQueueConnection** on page 214

CreateQueueConnectionFactory

Syntax

```
CreateQueueConnectionFactory(pczHost, iPort, iError, pczError)
```

Description

Constructs a **QueueConnectionFactory** object for the specified host and port. Once constructed, it can create **QueueConnection** objects for a point-to-point JMS provider.

Parameters

Name	Type	Description
pczHost	char*	Pointer to the text of the host name.
iPort	int	Port number.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

SBYN_QueueConnectionFactory*
Pointer to the **QueueConnectionFactory** object that was created.

CreateQueueConnection

Syntax

```
CreateQueueConnection(pQueCxnFac, iError, pczError)
```

Description

Constructs a **Connection** object; see **"The Connection Interface" on page 214**.

Parameters

Name	Type	Description
pQueCxnFac	SBYN_QueueConnectionFactory*	Pointer to the **QueueConnectionFactory** object creating the QueueConnection.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

SBYN_Connection* pointer.

8.1.10. The Connection Interface

A **Connection** object is an active connection to a JMS point-to-point provider or an active connection to a JMS pub/sub provider. A client uses a Connection to create one or more Sessions for producing and consuming messages.

The **Connection** interface includes the following methods:

ConnectionClose

Syntax

```
ConnectionClose(pConn, iError, pczError)
```

Description

Closes the specified connection.

Parameters

Name	Type	Description
pConnn	SBYN_Connection*	Pointer to the **Connection** object.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

None.

ConnectionGetClientID

Syntax

```
ConnectionGetClientID(pConn, iError, pczError)
```

Description

Retrieves the client ID associated with the specified **Connection** object.

Parameters

Name	Type	Description
pConn	SBYN_Connection*	Pointer to the **Connection** object.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

WString*
Pointer to a **WString** (wide string) object containing the text.

ConnectionSetClientID

Syntax

```
ConnectionSetClientID(pConn, pczClientID, iError, pczError)
```

Description

Sets the client ID to be associated with the specified **Connection** object. In other words, this method allows you to name the connection.

Parameters

Name	Type	Description
pConn	SBYN_Connection*	Pointer to the **Connection** object.
pczClientID	char*	Pointer to the text string for the client ID.

Name	Type	Description
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

None.

ConnectionStart

Syntax

```
ConnectionStart(pConn, iError, pczError)
```

Description

Starts (or restarts) delivering incoming messages via the specified **Connection** object. If the connection is already started, the call is ignored without error.

Parameters

Name	Type	Description
pConn	SBYN_Connection*	Pointer to the **Connection** object.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

None.

ConnectionStop

Syntax

```
ConnectionStop(pConn, iError, pczError)
```

Description

Temporarily halts delivering incoming messages via the specified **Connection** object. If the connection is already stopped, the call is ignored without error.

Parameters

Name	Type	Description
pConn	SBYN_Connection*	Pointer to the **Connection** object.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

None.

ConnectionCreateQueueSession

Syntax

```
ConnectionCreateQueueSession(pConn, fTransacted, iAckMode,
                             iError, pczError)
```

Description

Creates a **Session** object; see **"The Session Interface" on page 218**.

Parameters

Name	Type	Description
pConn	SBYN_Connection*	Pointer to the **Connection** object.
fTransacted	SBYN_BOOL	Flag: If **true**, the session is transacted.
iAckMode	int	Ignored if the session is transacted. If the session is not transacted, indicates whether the consumer or client acknowledges messages that it receives; see **Session Constants** on page 176.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

SBYN_Session* pointer.

ConnectionCreateTopicSession

Syntax

```
ConnectionCreateTopicSession(pCxn, fTransacted, iAckMode,
                             iError, pczError)
```

Description

Creates a **TopicSession** object.

Parameters

Name	Type	Description
pCxn	SBYN_Connection*	Pointer to the **Connection** object.
fTransacted	SBYN_BOOL	Flag: If **true**, the session is transacted

Name	Type	Description
iAckMode	int	Ignored if the session is transacted. If the session is not transacted, indicates whether the consumer or client acknowledges messages that it receives; see **Session Constants** on page 176.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

SBYN_Session* pointer.

8.1.11. The Session Interface

The Session interface includes the following methods:

- **SessionClose** on page 219
- **SessionCommit** on page 219
- **SessionGetTransacted** on page 219
- **SessionRecover** on page 220
- **SessionRollback** on page 220
- **SessionCreateBytesMessage** on page 221
- **SessionCreateTextMessage** on page 221
- **SessionCreateTextMessageEx** on page 222
- **SessionCreateQueue** on page 222
- **SessionCreateReceiver** on page 223
- **SessionCreateReceiveMessageSelector** on page 223
- **SessionCreateSender** on page 224
- **SessionCreateTemporaryQueue** on page 224
- **SessionCreateDurableSubscriber** on page 225
- **SessionCreateDurableSubscriberMessageSelector** on page 226
- **SessionCreatePublisher** on page 226
- **SessionCreateSubscriber** on page 227
- **SessionCreateSubscriberMessageSelector** on page 227
- **SessionCreateTemporaryTopic** on page 228
- **SessionCreateTopic** on page 229
- **SessionUnsubscribe** on page 229

SessionClose

Syntax

```
SessionClose(pSessn, iError, pczError)
```

Description

Closes the specified session.

Note: *Sessions should be closed when they are no longer needed.*

Parameters

Name	Type	Description
pSessn	SBYN_Session*	Pointer to the **Session** object.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

None.

SessionCommit

Syntax

```
SessionCommit(pSessn, iError, pczError)
```

Description

Commits all messages done in this transaction and releases any locks currently held.

Parameters

Name	Type	Description
pSessn	SBYN_Session*	Pointer to the **Session** object.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

None.

SessionGetTransacted

Syntax

```
SessionGetTransacted(pSessn, iError, pczError)
```

Description

Queries whether the specified session is or is not transacted.

Parameters

Name	Type	Description
pSessn	SBYN_Session*	Pointer to the **Session** object.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

SBYN_BOOL
Returns **true** if the session is transacted; otherwise, returns **false**.

SessionRecover

Syntax

```
SessionRecover(pSessn, iError, pczError)
```

Description

Stops message delivery in the specified session, causes all messages that might have been delivered but not acknowledged to be marked as **redelivered**, and restarts message delivery with the oldest unacknowledged message. Note that redelivered messages need not be delivered in the exact order they were originally delivered.

Parameters

Name	Type	Description
pSessn	SBYN_Session*	Pointer to the **Session** object.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

None.

SessionRollback

Syntax

```
SessionRollback(pSessn, iError, pczError)
```

Description

Rolls back any messages done in this transaction and releases any locks currently held.

Parameters

Name	Type	Description
pSessn	SBYN_Session*	Pointer to the **Session** object.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

None.

SessionCreateBytesMessage

Syntax

```
SessionCreateBytesMessage(pSessn, iError, pczError)
```

Description

Creates a **BytesMessage**— an object used to send a message containing a stream of uninterpreted bytes.

Parameters

Name	Type	Description
pSessn	SBYN_Session*	Pointer to the **Session** object.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

SBYN_Message*
Pointer to the created message object.

SessionCreateTextMessage

Syntax

```
SessionCreateTextMessage(pSessn, iError, pczError)
```

Description

Creates an uninitialized **TextMessage**— an object used to send a message containing a string to be supplied. Also see **"SessionCreateTextMessageEx" on page 222**.

Parameters

Name	Type	Description
pSessn	SBYN_Session*	Pointer to the **Session** object.

Name	Type	Description
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

SBYN_Message*
Pointer to the created message object.

SessionCreateTextMessageEx

Syntax

```
SessionCreateTextMessageEx(pSessn, pczText, iError, pczError)
```

Description

Creates an initialized **TextMessage**— an object used to send a message containing the supplied string. Also see **"SessionCreateTextMessage" on page 221**.

Parameters

Name	Type	Description
pQueSessn	SBYN_QueueSession*	Pointer to the **QueueSession** object.
pczText	char*	Pointer to the text string with which to initialize the message.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

SBYN_Message*
Pointer to the created message object.

SessionCreateQueue

Syntax

```
SessionCreateQueue(pSessn, pczQueName, iError, pczError)
```

Description

Creates an identity with a specific queue name; does not create a physical queue.

This functionality is provided for rare cases where clients need to dynamically create a queue identity with a provider-specific name. Clients that depend on this functionality are not portable.

To create a physical session, see **"SessionCreateTemporaryQueue" on page 224**.

Parameters

Name	Type	Description
pSessn	SBYN_Session*	Pointer to the **Session** object.
pczQueName	char*	Pointer to the name of the queue.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

SBYN_Destination* pointer.

SessionCreateReceiver

Syntax

```
SessionCreateReceiver(pSessn, pDest, iError, pczError)
```

Description

Creates a **Receiver** object to receive messages;

Parameters

Name	Type	Description
pSessn	SBYN_Session*	Pointer to the **Session** object.
pDest	SBYN_Destination*	Pointer to the queue.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

SBYN_Receiver* pointer.

SessionCreateReceiveMessageSelector

Syntax

```
SessionCreateReceiveMessageSelector(pSessn, pDest,
                                    pczSelector, iError, pczError)
```

Description

Creates a **Receiver** object to receive messages using a message selector. Also see **"SessionCreateReceiver" on page 223**.

Parameters

Name	Type	Description
pSessn	SBYN_Session*	Pointer to the object.
pDest	SBYN_Destination*	Pointer to the queue.
pczSelector	char*	Pointer to the text of the message selector.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

SBYN_QueueReceiver* pointer.

SessionCreateSender

Syntax

```
SessionCreateSender(pSessn, pDest, iError, pczError)
```

Description

Creates a **Sender** object to send messages.

Parameters

Name	Type	Description
pSessn	SBYN_Session*	Pointer to the **Session** object.
pDest	SBYN_Destination*	Pointer to the queue.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

SBYN_Sender* pointer.

SessionCreateTemporaryQueue

Syntax

```
SessionCreateTemporary(pSessn, iError, pczError)
```

Description

Creates a **Temporary** object for a specified session.

Parameters

Name	Type	Description
pSessn	SBYN_Session*	Pointer to the **Session** object.

Name	Type	Description
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

SBYN_Destination* pointer.

SessionCreateDurableSubscriber

Syntax

```
SessionCreateDurableSubscriber(pSessn, pDest, pczName,
                               iError, pczError)
```

Description

Creates a durable subscriber to the specified topic, specifying whether messages published by its own connection should be delivered to it.

Using pub/sub messaging, if a client needs to receive all the messages published on a topic, including messages published while the subscriber is inactive, it uses a *durable subscriber*. The JMS provider retains a record of this durable subscription and ensures that each message from the topic's publishers is retained until either it has been acknowledged by this durable subscriber or else it has expired.

Sessions with durable subscribers must always provide the same client ID, and each client must specify a name that (within the given client ID) uniquely identifies each durable subscription it creates. Only one session at a time can have a TopicSubscriber for a particular durable subscription. An *inactive* durable subscriber is one that exists but does not currently have a message consumer associated with it.

A client can change an existing durable subscription by creating a durable TopicSubscriber with the same name and a new topic (and/or *message selector*). Changing a durable subscriber is equivalent to deleting the old one and creating a new one.

Parameters

Name	Type	Description
pSessn	SBYN_Session*	Pointer to the **Session** object.
pDest	SBYN_Destination*	Pointer to the topic.
pczName	char*	Pointer to the text string containing the client ID of the durable subscriber.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

SBYN_TopicSubscriber*
Pointer to the created **TopicSubscriber** object.

SessionCreateDurableSubscriberMessageSelector

Syntax

```
TopicSessionCreateDurableSubscriberMessageSelector(pSessn, pDest,
                        pDest, pczName, pczSelector, iError, pczError)
```

Description

Creates a durable subscriber to the specified topic, using a message selector (*pczSelector*) and/or specifying whether messages published by its own connection should be delivered to it (*fNoLocal*).

Parameters

Name	Type	Description
pSessn	SBYN_Session*	Pointer to the **Session** object.
pDest	SBYN_Destination*	Pointer to the **Destination** object.
pczName	char*	Pointer to the text string containing the client ID of the durable subscriber.
pczSelector	char*	Pointer to the string containing the text of the message selector; only messages with properties matching the message selector expression are delivered. A value of null or an empty string indicates that there is no message selector for the message consumer.
fNoLocal	SBYN_BOOL	Flag, default **false**. If **true**, inhibits the delivery of messages published by its own connection.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

SBYN_TopicSubscriber*
Pointer to the created **TopicSubscriber** object.

SessionCreatePublisher

Syntax

```
TopicSessionCreatePublisher(pSessn, pDest, iError, pczError)
```

Description

Creates a publisher for the specified topic.

Parameters

Name	Type	Description
pSessn	SBYN_Session*	Pointer to the **Session** object.

Name	Type	Description
pDest	SBYN_Destination*	Pointer to the **Destination** object.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

SBYN_TopicPublisher*
Pointer to the created **TopicPublisher** object.

SessionCreateSubscriber

Syntax

```
SessionCreateSubscriber(pSessn, pDest, iError, pczError)
```

Description

Creates a *nondurable* subscriber to the specified topic—in other words, a subscriber that receives only those messages that are published while the subscriber is active.

Parameters

Name	Type	Description
pSessn	SBYN_Session*	Pointer to the **Session** object.
pDest	SBYN_Destination*	Pointer to the **Destination** object.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

SBYN_TopicSubscriber*
Pointer to the **TopicSubscriber** object.

SessionCreateSubscriberMessageSelector

Syntax

```
SessionCreateSubscriberMessageSelector(pSessn, pDest,
                    pczSelector, fNoLocal, iError, pczError)
```

Description

Creates a *nondurable* subscriber to the specified topic—in other words, a subscriber that receives only those messages that are published while the subscriber is active

In some cases, a connection may both publish and subscribe to a topic. The NoLocal parameter allows a subscriber to inhibit the delivery of messages published by its own connection. The default value for this attribute is **false**.

Parameters

Name	Type	Description
pSessn	SBYN_Session*	Pointer to the **Session** object.
pDest	SBYN_Destination*	Pointer to the **Destination** object.
pczSelector	char*	Pointer to the string containing the text of the message selector; only messages with properties matching the message selector expression are delivered. A value of null or an empty string indicates that there is no message selector for the message consumer.
fNoLocal	SBYN_BOOL	Flag, default **false**. If **true**, inhibits the delivery of messages published by its own connection.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

SBYN_TopicSubscriber*
Pointer to the **TopicSubscriber** object.

SessionCreateTemporaryTopic

Syntax

```
SessionTopicSessionCreateTemporaryTopic(pSessn,
                                        iError, pczError)
```

Description

Creates a temporary topic that lives only as long as the specified TopicConnection does (unless the topic is deleted earlier).

Parameters

Name	Type	Description
pSessn	SBYN_Session*	Pointer to the **Session** object.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

SBYN_Destination*
Pointer to the created **TemporaryTopic** object.

SessionCreateTopic

Syntax

```
SessionCreateTopic(pSessn, pczTopName, iError, pczError)
```

Description

Creates a topic identity with a specific topic name; does ***not*** create a physical topic.

This functionality is provided for rare cases where clients need to dynamically create a topic identity with a provider-specific name. Clients that depend on this functionality are not portable.

Parameters

Name	Type	Description
pSessn	SBYN_Session*	Pointer to the **Session** object.
pczTopName	char*	Pointer to the text string containing the name of the topic.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

SBYN_Destination*
Pointer to the created **Destination** object.

SessionUnsubscribe

Syntax

```
SessionUnsubscribe(pSessn, pczName, iError, pczError)
```

Description

Unsubscribes a durable subscription that has been created by a client. Note that it is an error to delete a durable subscription while there is an active TopicSubscriber for the subscription, or while a consumed message is part of a pending transaction or has not been acknowledged in the session.

Parameters

Name	Type	Description
pSessn	SBYN_Session*	Pointer to the **Session** object.
pczName	char*	Pointer to the text string containing the name used to identify this subscription.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

None.

8.1.12. The TopicConnectionFactory Interface

Using pub/sub messaging, a client uses a **TopicConnectionFactory** object to create **TopicConnection** objects.

The **TopicConnectionFactory** interface includes the following methods:

- **CreateTopicConnectionFactory** on page 230
- **CreateTopicConnection** on page 230

CreateTopicConnectionFactory

Syntax

```
CreateTopicConnectionFactory(pczHost, iPort, iError, pczError)
```

Description

Constructs a **TopicConnectionFactory** for the specified host and port. Once constructed, it can create **TopicConnection** objects for a pub/sub JMS provider.

Parameters

Name	Type	Description
pczHost	char*	Pointer to the text of the host name.
iPort	int	Port number.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

SBYN_TopicConnectionFactory*
Pointer to the **TopicConnectionFactory** object that was created.

CreateTopicConnection

Syntax

```
CreateTopicConnection(pTopCxnFac, iError, pczError)
```

Description

Constructs a **TopicConnection** object.

Parameters

Name	Type	Description
pTopCxnFac	SBYN_TopicConnectionFactory*	Pointer to the **TopicConnectionFactory** object creating the TopicConnection.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

SBYN_TopicConnection*
Pointer to the **TopicConnection** object that was created.

8.1.13. The Destination Interface

A **Destination** object encapsulates an address for a destination provided by SeeBeyond JMS. It can also include other data, such as configuration information or metadata.

The **Destination** interface includes the following methods:

- **GetDestinationName** on page 231
- **SetDestinationName** on page 232
- **DestinationToString** on page 232
- **DeleteDestination** on page 232

GetDestinationName

Syntax

```
GetDestinationName(pDest, iError, pczError)
```

Description

Retrieves the name of the specified **Destination** object.

Parameters

Name	Type	Description
pDest	SBYN_Destination*	Pointer to the **Destination** object.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

WString*
Pointer to a **WString** (wide string) object containing the text.

SetDestinationName

Syntax

```
SetDestinationName(pDest, pczName, iError, pczError)
```

Description

Sets the name of the specified **Destination** object.

Parameters

Name	Type	Description
pDest	SBYN_Destination*	Pointer to the **Destination** object.
pczName	char*	Pointer to the text string containing the name of the destination.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

None.

DestinationToString

Syntax

```
DestinationToString(pDest, iError, pczError)
```

Description

Retrieves a text string representation of the specified **Destination** object.

Parameters

Name	Type	Description
pDest	SBYN_Destination*	Pointer to the **Destination** object.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

WString*
Pointer to a **WString** (wide string) object containing the text.

DeleteDestination

Syntax

```
DeleteDestination(pDest, iError, pczError)
```

Description

Deletes the specified **Destination** object.

Parameters

Name	Type	Description
pDest	SBYN_Destination*	Pointer to the **Destination** object.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

None.

8.1.14. The QueueReceiver Interface

Using point-to-point messaging, a client uses a **QueueReceiver** object to receive messages that have been delivered to a queue.

The **QueueReceiver** interface includes the following methods:

- **QueueReceiverClose** on page 233
- **QueueReceiverGetMessageSelector** on page 234
- **QueueReceiverReceive** on page 234
- **QueueReceiverReceiveTimeOut** on page 235
- **QueueReceiverReceiveNoWait** on page 235
- **QueueReceiverGetQueue** on page 235

QueueReceiverClose

Syntax

```
QueueReceiverClose(pQueRecvr, iError, pczError)
```

Description

Closes the specified queue receiver.

Note: *When a message consumer is no longer needed, it should be closed.*

Parameters

Name	Type	Description
pQueRecvr	SBYN_QueueReceiver*	Pointer to the **QueueReceiver** object.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

None.

QueueReceiverGetMessageSelector

Syntax

```
QueueReceiverGetMessageSelector(pQueRecvr, iError, pczError)
```

Description

Retrieves the message selector expression associated with this queue receiver.

Parameters

Name	Type	Description
pQueRecvr	SBYN_QueueReceiver*	Pointer to the **QueueReceiver** object.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

char*
Pointer to the string containing the message selector text. Returns null if no message selector exists, or if the message selector was set to null or the empty string.

QueueReceiverReceive

Syntax

```
QueueReceiverReceive(pQueRecvr, iError, pczError)
```

Description

Receives the next message produced for this queue. If the called within a transaction, the queue retains the message until the transaction commits.

Parameters

Name	Type	Description
pQueRecvr	SBYN_QueueReceiver*	Pointer to the **QueueReceiver** object.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

SBYN_Message*
Pointer to the received message.

QueueReceiverReceiveTimeOut

Syntax

```
QueueReceiverReceiveTimeOut(pQueRecvr, lTimeout, iError, pczError)
```

Description

Receives the next message that arrives within the specified timeout interval. A timeout value of 0 makes this function equivalent to **"QueueReceiverReceive" on page 234**.

Parameters

Name	Type	Description
pQueRecvr	SBYN_QueueReceiver*	Pointer to the **QueueReceiver** object.
lTimeout	long	The timeout value.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

SBYN_Message*
Pointer to the received message, or null if no message is available.

QueueReceiverReceiveNoWait

Syntax

```
QueueReceiverReceiveNoWait(pQueRecvr, iError, pczError)
```

Description

Receives the next message produced for this queue if one is immediately available.

Parameters

Name	Type	Description
pQueRecvr	SBYN_QueueReceiver*	Pointer to the **QueueReceiver** object.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

SBYN_Message*
Pointer to the received message, or null if no message is available.

QueueReceiverGetQueue

Syntax

```
QueueReceiverGetQueue(pQueRecvr, iError, pczError)
```

Description

Retrieves the queue associated with the specified queue receiver.

Parameters

Name	Type	Description
pQueRecvr	SBYN_QueueReceiver*	Pointer to the **QueueReceiver** object.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

SBYN_Destination*
Pointer to the queue.

8.1.15. The TopicSubscriber Interface

The TopicSubscriber interface includes the following methods:

TopicSubscriberClose

Syntax

```
TopicSubscriberClose(pTopSub, iError, pczError)
```

Description

Closes the specified topic subscriber.

Note: *When a message consumer is no longer needed, it should be closed.*

Parameters

Name	Type	Description
pTopSub	SBYN_TopicSubscriber*	Pointer to the **TopicSubscriber** object.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

None.

TopicSubscriberGetMessageSelector

Syntax

```
TopicSubscriberGetMessageSelector(pTopSub, iError, pczError)
```

Description

Retrieves the message selector expression associated with the specified topic subscriber.

Parameters

Name	Type	Description
pTopSub	SBYN_TopicSubscriber*	Pointer to the **TopicSubscriber** object.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

char*
Pointer to the string containing the message selector text. Returns null if no message selector exists, or if the message selector was set to null or the empty string.

TopicSubscriberGetNoLocal

Syntax

```
TopicSubscriberGetNoLocal(pTopSub, iError, pczError)
```

Description

Queries whether the NoLocal flag is set for the specified topic subscriber.

Parameters

Name	Type	Description
pTopSub	SBYN_TopicSubscriber*	Pointer to the **TopicSubscriber** object.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

SBYN_BOOL
If **false** (the default), the subscriber can also publish via the same connection.
If **true**, delivery of messages published via its own connection is inhibited.

TopicSubscriberGetTopic

Syntax

```
TopicSubscriberGetTopic(pTopSub, iError, pczError)
```

Description

Retrieves the topic associated with the specified topic subscriber.

Parameters

Name	Type	Description
pTopSub	SBYN_TopicSubscriber*	Pointer to the **TopicSubscriber** object.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

SBYN_Destination*
Pointer to the topic.

TopicSubscriberReceive

Syntax

```
TopicSubscriberReceive(pTopSub, iError, pczError)
```

Description

Receives the next message produced for this topic. If called within a transaction, the topic retains the message until the transaction commits.

Parameters

Name	Type	Description
pTopSub	SBYN_TopicSubscriber*	Pointer to the **TopicSubscriber** object.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

SBYN_Message*
Pointer to the received message.

TopicSubscriberReceiveTimeout

Syntax

```
TopicSubscriberReceiveTimeOut(pTopSub, lTimeout, iError, pczError)
```

Description

Receives the next message that arrives within the specified timeout interval. A timeout value of 0 makes this function equivalent to **"TopicSubscriberReceive" on page 238**.

Parameters

Name	Type	Description
pTopSub	SBYN_TopicSubscriber*	Pointer to the **TopicSubscriber** object.
lTimeout	long	The timeout value
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

SBYN_Message*
Pointer to the received message, or null if no message is available.

TopicSubscriberReceiveNoWait

Syntax

```
TopicSubscriberReceiveNoWait(pTopSub, iError, pczError)
```

Description

Receives the next message produced for this topic if one is immediately available.

Parameters

Name	Type	Description
pTopSub	SBYN_TopicSubscriber*	Pointer to the **TopicSubscriber** object.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

SBYN_Message*
Pointer to the received message, or null if no message is available.

Syntax

```
TopicSubscriberReceiveNoWait(pTopSub, iError, pczError)
```

8.1.16. The QueueSender Interface

Using point-to-point messaging, a client uses a **QueueSender** object to send messages to a queue. After sending a message to a queue, a client may retain the message and modify it without affecting the message that has been sent. The same message object may be sent multiple times.

The **QueueSender** interface includes the following methods:

QueueSenderClose

Syntax

```
QueueSenderClose(pQueSender, iError, pczError)
```

Description

Closes the specified queue sender.

Note: *When a message producer is no longer needed, it should be closed.*

Parameters

Name	Type	Description
pQueSender	SBYN_QueueSender*	Pointer to the **QueueSender** object.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

None.

QueueSenderGetDeliveryMode

Syntax

```
QueueSenderGetDeliveryMode(pQueSender, iError, pczError)
```

Description

Retrieves the value of the **DeliveryMode** property of the specified queue sender.

Parameters

Name	Type	Description
pQueSender	SBYN_QueueSender*	Pointer to the **QueueSender** object.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

int

See **"DeliveryMode Constants" on page 176**.

QueueSenderGetDisableMessageID

Syntax

```
QueueSenderGetDisableMessageID(pQueSender, iError, pczError)
```

Description

Queries whether message IDs are or are not disabled for the specified queue sender.

Parameters

Name	Type	Description
pQueSender	SBYN_QueueSender*	Pointer to the **QueueSender** object.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

SBYN_BOOL

Returns **true** if message IDs are disabled; otherwise, returns **false**.

QueueSenderGetDisableMessageTimestamp

Syntax

```
QueueSenderGetDisableMessageTimestamp(pQueSender, iError, pczError)
```

Description

Queries whether message timestamping is or is not disabled for the specified queue sender.

Parameters

Name	Type	Description
pQueSender	SBYN_QueueSender*	Pointer to the **QueueSender** object.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

SBYN_BOOL
Returns **true** if message timestamping is disabled; otherwise, returns **false**.

QueueSenderGetJMS_ProducerID

Syntax

```
QueueSenderGetJMS_ProducerID(pQueSender, iError, pczError)
```

Description

Retrieves the value of the **ProducerID** property for the specified queue sender.

Parameters

Name	Type	Description
pQueSender	SBYN_QueueSender*	Pointer to the **QueueSender** object.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

WString*
Pointer to a **WString** (wide string) object containing the text.

QueueSenderGetPriority

Syntax

```
QueueSenderGetPriority(pQueSender, iError, pczError)
```

Description

Queries the value of the message **Priority** property of the specified queue sender.

Parameters

Name	Type	Description
pQueSender	SBYN_QueueSender*	Pointer to the **QueueSender** object.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

int
Message priority level, from **0** (least expedited) through **9** (most expedited). See **"Miscellaneous Constants Setting Message Class Defaults" on page 178**.

QueueSenderGetQueue

Syntax

```
QueueSenderGetQueue(pQueSender, iError, pczError)
```

Description

Retrieves the queue associated with the queue sender.

Parameters

Name	Type	Description
pQueSender	SBYN_QueueSender*	Pointer to the **QueueSender** object.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

SBYN_Destination*
Pointer to the queue.

QueueSenderGetTimeToLive

Syntax

```
QueueSenderGetTimeToLive(pQueSender, iError, pczError)
```

Description

Queries the value of the **TimeToLive** property of the specified queue sender.

Parameters

Name	Type	Description
pQueSender	SBYN_QueueSender*	Pointer to the **QueueSender** object.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

long
Length of time (milliseconds elapsed since dispatch time) that a produced message should be retained by JMS. See **"Miscellaneous Constants Setting Message Class Defaults" on page 178**.

QueueSenderSend

Syntax

```
QueueSenderSend(pQueSender, pMsg, iError, pczError)
```

Description

Sends the specified message to the queue, using default values for properties (**DeliveryMode**, **Priority**, and **TimeToLive**) of the specified queue sender. If you need to override the default values, see **"QueueSenderSendEx" on page 244**.

To specify the queue destination, see **"QueueSenderSendToQueue" on page 245**.

Parameters

Name	Type	Description
pQueSender	SBYN_QueueSender*	Pointer to the **QueueSender** object.
pMsg	SBYN_Message*	Pointer to the message to send.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

None.

QueueSenderSendEx

Syntax

```
QueueSenderSendEx(pQueSender, pMsg,
                  iDelivMode, iPriority, lMilSecToLive,
                  iError, pczError)
```

Description

Sends the specified message to the queue, overriding one or more default values for properties of the specified queue sender. Compare to **"QueueSenderSend" on page 244**.

To specify the queue destination, see **"QueueSenderSendToQueueEx" on page 246**.

Parameters

Name	Type	Description
pQueSender	SBYN_QueueSender*	Pointer to the **QueueSender** object.
pMsg	SBYN_Message*	Pointer to the message to send.
iDelivMode	int	Value to be used for **DeliveryMode**; see **"DeliveryMode Constants" on page 176**.
iPriority	int	Value to be used for **Priority**; see **"Miscellaneous Constants Setting Message Class Defaults" on page 178**.
lMilSecToLive	long	Value to be used for **TimeToLive**: Length of time (milliseconds elapsed since dispatch time) that a produced message should be retained by JMS. See **"Miscellaneous Constants Setting Message Class Defaults" on page 178**.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

None.

QueueSenderSendToQueue

Syntax

```
QueueSenderSendToQueue(pQueSender, pDest, pMsg, iError, pczError)
```

Description

Sends the specified message to the specified queue, using default values for properties (**DeliveryMode**, **Priority**, and **TimeToLive**) of the specified queue sender. Compare to **"QueueSenderSendToQueueEx" on page 246**.

Typically, a message producer is assigned a queue at creation time; however, the JMS API also supports unidentified message producers, which require that the queue be supplied every time a message is sent. Compare to **"QueueSenderSend" on page 244**.

Parameters

Name	Type	Description
pQueSender	SBYN_QueueSender*	Pointer to the **QueueSender** object.

Name	Type	Description
pDest	SBYN_Destination*	Pointer to the queue.
pMsg	SBYN_Message*	Pointer to the message to send.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

None.

QueueSenderSendToQueueEx

Syntax

```
QueueSenderSendToQueueEx(pQueSender, pDest, pMsg,
                         iDelivMode, iPriority, lMilSecToLive,
                         iError, pczError)
```

Description

Sends the specified message to the specified queue, overriding one or more default values for properties of the specified queue sender. Compare to **"QueueSenderSendToQueue" on page 245**.

Parameters

Name	Type	Description
pQueSender	SBYN_QueueSender*	Pointer to the **QueueSender** object.
pDest	SBYN_Destination*	Pointer to the queue.
pMsg	SBYN_Message*	Pointer to the message to send.
iDelivMode	int	Value to be used for **DeliveryMode**; see **"DeliveryMode Constants" on page 176**.
iPriority	int	Value to be used for **Priority**; see **"Miscellaneous Constants Setting Message Class Defaults" on page 178**.
lMilSecToLive	long	Value to be used for **TimeToLive**: Length of time (milliseconds elapsed since dispatch time) that a produced message should be retained by JMS. See **"Miscellaneous Constants Setting Message Class Defaults" on page 178**.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

None.

QueueSenderSetDeliveryMode

Syntax

```
QueueSenderSetDeliveryMode(pQueSender, iDelivMode, iError, pczError)
```

Description

Sets the value of the **DeliveryMode** property of the specified queue sender. See **"DeliveryMode Constants" on page 176**.

Parameters

Name	Type	Description
pQueSender	SBYN_QueueSender*	Pointer to the **QueueSender** object.
iDelivMode	int	Value for the **DeliveryMode** property.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

None.

QueueSenderSetDisableMessageID

Syntax

```
QueueSenderSetDisableMessageID(pQueSender, fDisabled,
                               iError, pczError)
```

Description

Determines whether message IDs are disabled for this queue sender. Default **false**.

Parameters

Name	Type	Description
pQueSender	SBYN_QueueSender*	Pointer to the **QueueSender** object.
fDisabled	SBYN_BOOL	Flag, default **false**; if **true**, message IDs are disabled.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

None.

QueueSenderSetDisableMessageTimestamp

Syntax

```
QueueSenderSetDisableMessageTimestamp(pQueSender, fDisabled,
                                      iError, pczError)
```

Description

Determines whether message timestamping is disabled for this queue sender. Default **false**.

Since message timestamps take effort to create and increase the size of a message, this flag can be set **true** to reduce overhead if message IDs are not used by an application.

Parameters

Name	Type	Description
pQueSender	SBYN_QueueSender*	Pointer to the **QueueSender** object.
fDisabled	SBYN_BOOL	Flag, default **false**; if **true**, message timestamping is disabled.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

None.

QueueSenderSetJMS_ProducerID

Syntax

```
QueueSenderSetJMS_ProducerID(pQueSender, pczProdID, iError, pczError)
```

Description

Sets the value of the **ProducerID** property for the specified queue sender.

Parameters

Name	Type	Description
pQueSender	SBYN_QueueSender*	Pointer to the **QueueSender** object.
pczProdID	char*	Pointer to text string containing the Producer ID.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

None.

QueueSenderSetPriority

Syntax

```
QueueSenderSetPriority(pQueSender, iPriority, iError, pczError)
```

Description

Sets the value of the message **Priority** property, from **0** (least expedited) through **9** (most expedited). See **"Miscellaneous Constants Setting Message Class Defaults" on page 178**.

Parameters

Name	Type	Description
pQueSender	SBYN_QueueSender*	Pointer to the **QueueSender** object.
iPriority	int	Message priority level.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

None.

QueueSenderSetTimeToLive

Syntax

```
QueueSenderSetTimeToLive(pQueSender, lMilSecToLive, iError, pczError)
```

Description

Sets the value of the **TimeToLive** property of the specified queue sender.

Parameters

Name	Type	Description
pQueSender	SBYN_QueueSender*	Pointer to the **QueueSender** object.
lMilSecToLive	long	Value to be used for TimeToLive: Length of time (milliseconds elapsed since dispatch time) that a produced message should be retained by JMS. See **"Miscellaneous Constants Setting Message Class Defaults" on page 178**.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

None.

8.1.17. The TopicPublisher Interface

The TopicPublisher interface includes the following methods:

TopicPublisherClose

Syntax

```
TopicPublisherClose(pTopPub, iError, pczError)
```

Description

Closes the specified topic publisher.

Note: *When a message producer is no longer needed, it should be closed.*

Parameters

Name	Type	Description
pTopPub	SBYN_TopicPublisher*	Pointer to the **TopicPublisher** object.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

None.

TopicPublisherGetDeliveryMode

Syntax

```
TopicPublisherGetDeliveryMode(pTopPub, iError, pczError)
```

Description

Retrieves the value of the **DeliveryMode** property of the specified topic publisher.

Parameters

Name	Type	Description
pTopPub	SBYN_TopicPublisher*	Pointer to the **TopicPublisher** object.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

int

See **"DeliveryMode Constants" on page 176**.

TopicPublisherGetDisableMessageID

Syntax

```
TopicPublisherGetDisableMessageID(pTopPub, iError, pczError)
```

Description

Queries whether message IDs are or are not disabled for the specified topic publisher.

Parameters

Name	Type	Description
pTopPub	SBYN_TopicPublisher*	Pointer to the **TopicPublisher** object.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

SBYN_BOOL

Returns **true** if message IDs are disabled; otherwise, returns **false**.

TopicPublisherGetDisableMessageTimestamp

Syntax

```
TopicPublisherGetDisableMessageTimestamp(pTopPub, iError, pczError)
```

Description

Queries whether message IDs are or are not disabled for the specified topic publisher.

Parameters

Name	Type	Description
pTopPub	SBYN_TopicPublisher*	Pointer to the **TopicPublisher** object.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

int
Returns **true** if message timestamping is disabled; otherwise, returns **false**.

TopicPublisherGetJMS_ProducerID

Syntax

```
TopicPublisherGetJMS_ProducerID(pTopPub, iError, pczError)
```

Description

Retrieves the value of the **ProducerID** property for the specified topic publisher.

Parameters

Name	Type	Description
pTopPub	SBYN_TopicPublisher*	Pointer to the **TopicPublisher** object.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

WString*
Pointer to a **WString** (wide string) object containing the text.

TopicPublisherGetPriority

Syntax

```
TopicPublisherGetPriority(pTopPub, iError, pczError)
```

Description

Queries the value of the message **Priority** property of the specified topic publisher.

Parameters

Name	Type	Description
pTopPub	SBYN_TopicPublisher*	Pointer to the **TopicPublisher** object.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

int

Message priority level, from **0** (least expedited) through **9** (most expedited). See **"Miscellaneous Constants Setting Message Class Defaults" on page 178**.

TopicPublisherGetTimeToLive

Syntax

```
TopicPublisherGetTimeToLive(pTopPub, iError, pczError)
```

Description

Queries the value of the **TimeToLive** property of the specified topic publisher.

Parameters

Name	Type	Description
pTopPub	SBYN_TopicPublisher*	Pointer to the **TopicPublisher** object.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

long

Length of time (milliseconds elapsed since dispatch time) that a produced message should be retained by JMS. See **"Miscellaneous Constants Setting Message Class Defaults" on page 178**.

TopicPublisherGetTopic

Syntax

```
TopicPublisherGetTopic(pTopPub, iError, pczError)
```

Description

Retrieves the topic associated with the specified topic publisher.

Parameters

Name	Type	Description
pTopPub	SBYN_TopicPublisher*	Pointer to the **TopicPublisher** object.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

SBYN_Destination*
Pointer to the topic.

TopicPublisherPublish

Syntax

```
TopicPublisherPublish(pTopPub, pMsg, iError, pczError)
```

Description

Publishes the specified message to the topic, using default values for properties (**DeliveryMode**, **Priority**, and **TimeToLive**) of the specified topic publisher. If you need to override the default values, see **"TopicPublisherPublishEx" on page 254**.

To specify the topic destination, see **"TopicPublisherPublishToTopic" on page 255**.

Parameters

Name	Type	Description
pTopPub	SBYN_TopicPublisher*	Pointer to the **TopicPublisher** object.
pMsg	SBYN_Message*	Pointer to the message to be published.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

None.

TopicPublisherPublishEx

Syntax

```
TopicPublisherPublishEx(pTopPub, pMsg,
                        iDelivMode, iPriority, lMilSecToLive,
                        iError, pczError)
```

Description

Publishes the specified message to the topic, overriding one or more default values for properties of the specified topic publisher. Compare to **"TopicPublisherPublish" on page 254**.

To specify the topic destination, see "TopicPublisherPublishToTopicEx" on page 256.

Parameters

Name	Type	Description
pTopPub	SBYN_TopicPublisher*	Pointer to the **TopicPublisher** object.
pMsg	SBYN_Message*	Pointer to the message to be published.
iDelivMode	int	Value to be used for **DeliveryMode**; see "DeliveryMode Constants" on page 176.
iPriority	int	Value to be used for **Priority**; see "Miscellaneous Constants Setting Message Class Defaults" on page 178.
lMilSecToLive	long	Value to be used for **TimeToLive**: Length of time (milliseconds elapsed since dispatch time) that a produced message should be retained by JMS. See "Miscellaneous Constants Setting Message Class Defaults" on page 178.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

None.

TopicPublisherPublishToTopic

Syntax

```
TopicPublisherPublishToTopic(pTopPub, pDest, pMsg, iError, pczError)
```

Description

Publishes the specified message to the specified topic, using default values for properties (**DeliveryMode**, **Priority**, and **TimeToLive**) of the specified topic publisher. Compare to "TopicPublisherPublishToTopicEx" on page 256.

Typically, a message producer is assigned a topic at creation time; however, the JMS API also supports unidentified message producers, which require that the topic be supplied every time a message is sent. Compare to "TopicPublisherPublishEx" on page 254.

Parameters

Name	Type	Description
pTopPub	SBYN_TopicPublisher*	Pointer to the **TopicPublisher** object.
pDest	SBYN_Destination*	Pointer to the topic.
pMsg	SBYN_Message*	Pointer to the message.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

None.

TopicPublisherPublishToTopicEx

Syntax

```
TopicPublisherPublishToTopicEx(pTopPub, pDest, pMsg,
                               iDelivMode, iPriority, lMilSecToLive,
                               iError, pczError)
```

Description

Publishes the specified message to the specified topic, overriding one or more default values for properties of the specified topic publisher. Compare to **"TopicPublisherPublishToTopic" on page 255**.

Parameters

Name	Type	Description
pTopPub	SBYN_TopicPublisher*	Pointer to the **TopicPublisher** object.
pDest	SBYN_Destination*	Pointer to the topic.
pMsg	SBYN_Message*	Pointer to the message to publish.
iDelivMode	int	Value to be used for **DeliveryMode**; see **"DeliveryMode Constants" on page 176**.
iPriority	int	Value to be used for **Priority**; see **"Miscellaneous Constants Setting Message Class Defaults" on page 178**.
lMilSecToLive	long	Value to be used for **TimeToLive**: Length of time (milliseconds elapsed since dispatch time) that a produced message should be retained by JMS. See **"Miscellaneous Constants Setting Message Class Defaults" on page 178**.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

None.

TopicPublisherSetDeliveryMode

Syntax

```
TopicPublisherSetDeliveryMode(pTopPub, iDelivMode, iError, pczError)
```

Description

Sets the value of the **DeliveryMode** property of the specified topic publisher. See **"DeliveryMode Constants" on page 176**.

Parameters

Name	Type	Description
pTopPub	SBYN_TopicPublisher*	Pointer to the **TopicPublisher** object.
iDelivMode	int	Value for the **DeliveryMode** property.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

None.

TopicPublisherSetDisableMessageID

Syntax

```
TopicPublisherSetDisableMessageID(pTopPub, fDisabled,
                                  iError, pczError)
```

Description

Determines whether to disable message IDs for this topic publisher. Default **false**.

Parameters

Name	Type	Description
pTopPub	SBYN_TopicPublisher*	Pointer to the **TopicPublisher** object.
iDisable	SBYN_BOOL	Flag, default **false**; if **true**, message IDs are disabled.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

None.

TopicPublisherSetDisableMessageTimestamp

Syntax

```
TopicPublisherSetDisableMessageTimestamp(pTopPub, fDisabled,
                                         iError, pczError)
```

Description

Determines whether message timestamping is disabled for this topic publisher. Default **false**.

Since message timestamps take effort to create and increase the size of a message, this flag can be set **true** to reduce overhead if message IDs are not used by an application.

Parameters

Name	Type	Description
pTopPub	SBYN_TopicPublisher*	Pointer to the **TopicPublisher** object.
fDisabled	SBYN_BOOL	Flag, default **false**; if **true**, message timestamping is disabled.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

None.

TopicPublisherSetJMS_ProducerID

Syntax

```
TopicPublisherSetJMS_ProducerID(pTopPub, pczProdID, iError, pczError)
```

Description

Sets the value of the **ProducerID** property for the specified topic publisher.

Parameters

Name	Type	Description
pTopPub	SBYN_TopicPublisher*	Pointer to the **TopicPublisher** object.
pczProdID	char*	Pointer to the text string containing the Producer ID.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

None.

TopicPublisherSetPriority

Syntax

```
TopicPublisherSetPriority(pTopPub, iPriority09, iError, pczError)
```

Description

Sets the value of the message **Priority** property of the specified topic publisher.

Parameters

Name	Type	Description
pTopPub	SBYN_TopicPublisher*	Pointer to the **TopicPublisher** object.
iPriority09	int	Message priority, from 0 (least expedited) through 9 (most expedited). See **"Miscellaneous Constants Setting Message Class Defaults" on page 178**.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

None.

TopicPublisherSetTimeToLive

Syntax

```
TopicPublisherSetTimeToLive(pTopPub, lMSecToLive, iError, pczError)
```

Description

Sets the value of the **TimeToLive** property for the specified topic publisher.

Parameters

Name	Type	Description
pTopPub	SBYN_TopicPublisher*	Pointer to the **TopicPublisher** object.
lMilSecToLive	long	Length of time (milliseconds elapsed since dispatch time) that a produced message should be retained by JMS. See **"Miscellaneous Constants Setting Message Class Defaults" on page 178**.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

None.

8.1.18. The TopicRequestor Interface

The **TopicRequestor** object is used in request/reply messaging to simplify making service requests. Given a non-transacted **TopicSession** object and a destination **Topic**, it creates a **TemporaryTopic** object for the responses and provides a request method that sends the request message and waits for its reply.

The **TopicRequestor** interface includes the following methods:

CreateTopicRequestor

Syntax

```
CreateTopicRequestor(pTopSessn, pDest, iError, pczError)
```

Description

Constructs a **TopicRequestor** object. This implementation assumes that the parent topic session is non-transacted and that the value of its **DeliveryMode** property is either AUTO_ACKNOWLEDGE or DUPS_OK_ACKNOWLEDGE.

Parameters

Name	Type	Description
pTopSessn	SBYN_TopicSession*	Pointer to the parent **TopicSession** object.
pDest	SBYN_Destination*	Pointer to the destination topic.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

SBYN_TopicRequestor*
Pointer to the constructed **TopicRequestor** object.

TopicRequestorRequest

Syntax

```
TopicRequestorRequest(pTopReq, pMsg, iError, pczError)
```

Description

Sends a request and waits for a reply, without any upper limit on the time to wait (compare **"TopicRequestorRequestTimeout" on page 261**). The temporary topic is used for the **JMSReplyTo** destination; the first reply is returned, and any following replies are discarded.

Parameters

Name	Type	Description
pTopReq	SBYN_TopicRequestor*	Pointer to the **TopicRequestor** object.
pMsg	SBYN_Message*	Pointer to the message.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

SBYN_Message*
Pointer to the reply message.

TopicRequestorRequestTimeout

Syntax

```
TopicRequestorRequestTimeOut(pTopReq, pMsg, lTimeout,
                             iError, pczError)
```

Description

Sends a request and waits for a reply, but only during the specified period of time. Compare to **"TopicRequestorRequest" on page 260**.

Parameters

Name	Type	Description
pTopReq	SBYN_TopicRequestor*	Pointer to the **TopicRequestor** object.
pMsg	SBYN_Message*	Pointer to the message.
lTimeout	long	Timeout value, in milliseconds.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

SBYN_Message*
Pointer to the reply message, or **0** if no reply arrives before the waiting period ends.

TopicRequestorClose

Syntax

```
TopicRequestorClose(pTopReq, iError, pczError)
```

Description

Closes the specified topic requestor.

Note: *When a message producer or consumer is no longer needed, it should be closed.*

Parameters

Name	Type	Description
pTopReq	SBYN_TopicRequestor*	Pointer to the **TopicRequestor** object.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

None.

8.1.19. The QueueRequestor Interface

The **QueueRequestor** object is used in request/reply messaging to simplify making service requests. Given a non-transacted **QueueSession** object and a destination **Queue**, it creates a **TemporaryQueue** object for the responses and provides a request method that sends the request message and waits for its reply.

The **QueueRequestor** interface includes the following methods:

- **CreateQueueRequestor** on page 262
- **QueueRequestorClose** on page 263
- **QueueRequestorRequest** on page 263
- **QueueRequestorRequestTimeout** on page 264

CreateQueueRequestor

Syntax

```
CreateQueueRequestor(pQueSessn, pDest, iError, pczError)
```

Description

Constructs a **QueueRequestor** object. This implementation assumes that the parent queue session is non-transacted and that the value of its **DeliveryMode** property is either AUTO_ACKNOWLEDGE or DUPS_OK_ACKNOWLEDGE.

Parameters

Name	Type	Description
pQueSessn	SBYN_QueueSession*	Pointer to the parent **QueueSession** object.
pDest	SBYN_Destination*	Pointer to the destination queue.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

SBYN_QueueRequestor*
Pointer to the constructed **QueueRequestor** object.

QueueRequestorClose

Syntax

```
QueueRequestorClose(pQueReq, iError, pczError)
```

Description

Closes the specified queue requestor.

Note: *When a message producer or consumer is no longer needed, it should be closed.*

Parameters

Name	Type	Description
pQueReq	SBYN_QueueRequestor*	Pointer to the **QueueRequestor** object.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

None.

QueueRequestorRequest

Syntax

```
QueueRequestorRequest(pQueReq, pMsg, iError, pczError)
```

Description

Sends a request and waits for a reply, without any upper limit on the time to wait (compare **"QueueRequestorRequestTimeout" on page 264**). The temporary queue is used for the **JMSReplyTo** destination. Only one reply per request is expected.

Parameters

Name	Type	Description
pTopReq	SBYN_TopicRequestor*	Pointer to the **TopicRequestor** object.
pMsg	SBYN_Message*	Pointer to the message.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

SBYN_Message*
Pointer to the reply message.

QueueRequestorRequestTimeout

Syntax

```
QueueRequestorRequestTimeOut(pQueReq, pMsg, lTimeout,
                             iError, pczError)
```

Description

Sends a request and waits for a reply, but only during the specified period of time. Compare to **"QueueRequestorRequest" on page 263**.

Parameters

Name	Type	Description
pQueReq	SBYN_QueueRequestor*	Pointer to the **QueueRequestor** object.
pMsg	SBYN_Message*	Pointer to the message.
lTimeout	long	Timeout value, in milliseconds.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

SBYN_Message*
Pointer to the reply message, or **0** if no reply arrives before the waiting period ends.

8.1.20. Destructor Methods

Destructor methods include the following:

DeleteQueueConnectionFactory

Syntax

```
DeleteQueueConnectionFactory(pQueCxnFac, iError, pczError)
```

Description

Deletes the specified queue connection factory.

Parameters

Name	Type	Description
pQueCxnFac	SBYN_QueueConnectionFactory*	Pointer to the **QueueConnectionFactory** object.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

None.

DeleteQueueConnection

Syntax

```
DeleteQueueConnection(pQueCxn, iError, pczError)
```

Description

Deletes the specified queue connection.

Parameters

Name	Type	Description
pQueCxn	SBYN_QueueConnection*	Pointer to the **QueueConnection** object.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

None.

DeleteQueueReceiver

Syntax

```
DeleteQueueReceiver(pQueRecvr, iError, pczError)
```

Description

Deletes the specified queue receiver.

Parameters

Name	Type	Description
pQueRecvr	SBYN_QueueReceiver*	Pointer to the **QueueReceiver** object.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

None.

DeleteQueueRequestor

Syntax

```
DeleteQueueRequestor(pQueReq, iError, pczError)
```

Description

Deletes the specified queue requestor.

Parameters

Name	Type	Description
pQueReq	SBYN_QueueRequestor *	Pointer to the **QueueRequestor** object.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

None.

DeleteQueueSender

Syntax

```
DeleteQueueSender(pQueSender, iError, pczError)
```

Description

Deletes the specified queue sender.

Parameters

Name	Type	Description
pQueSender	SBYN_QueueSender*	Pointer to the **QueueSender** object.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

None.

DeleteQueueSession

Syntax

```
DeleteQueueSession(pQueSessn, iError, pczError)
```

Description

Deletes the specified queue session.

Parameters

Name	Type	Description
pQueSessn	SBYN_QueueSession*	Pointer to the **QueueSession** object.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

None.

DeleteTopicConnectionFactory

Syntax

```
DeleteTopicConnectionFactory(pTopCxnFac, iError, pczError)
```

Description

Deletes the specified topic connection factory.

Parameters

Name	Type	Description
pTopCxnFac	SBYN_TopicConnectionFactory*	Pointer to the **TopicConnectionFactory** object.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

None.

DeleteTopicConnection

Syntax

```
DeleteTopicConnection(pTopCxn, iError, pczError)
```

Description

Deletes the specified topic connection.

Parameters

Name	Type	Description
pTopCxn	SBYN_TopicConnection*	Pointer to the **TopicConnection** object.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

None.

DeleteTopicSession

Syntax

```
DeleteTopicSession(pTopSession, iError, pczError)
```

Description

Deletes the specified topic session.

Parameters

Name	Type	Description
pTopSessn	SBYN_TopicSession*	Pointer to the **TopicSession** object.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

None.

DeleteTopicSubscriber

Syntax

```
DeleteTopicSubscriber(pTopSub, iError, pczError)
```

Description

Deletes the specified topic subscriber.

Parameters

Name	Type	Description
pTopSub	SBYN_TopicSubscriber*	Pointer to the **TopicSubscriber** object.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

None.

DeleteTopicRequestor

Syntax

```
DeleteTopicRequestor(pTopReq, iError, pczError)
```

Description

Deletes the specified topic requestor.

Parameters

Name	Type	Description
qTopReq	SBYN_TopicRequestor*	Pointer to the **TopicRequestor** object.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

None.

DeleteTopicPublisher

Syntax

```
DeleteTopicPublisher(pTopPub, iError, pczError)
```

Description

Deletes the specified topic publisher.

Parameters

Name	Type	Description
pTopPub	SBYN_TopicPublisher*	Pointer to the **TopicPublisher** object.
iError	int*	Pointer to the error number.

Name	Type	Description
pczError	char*	Pointer to the error message text.

Return Value

None.

DeleteMessage

Syntax

```
DeleteMessage(pMsg, iError, pczError)
```

Description

Deletes the specified message.

Parameters

Name	Type	Description
pMsg	SBYN_Message*	Pointer to the message.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

None.

8.1.21. The WString Helper Interface

The **WString** structure (wide string—not part of the standard JMS API), is used to facilitate the handling of text strings.

The **WString** helper interface includes the following methods:

- **CharToWString** on page 270
- **DeleteWString** on page 271
- **WStringToChar** on page 271

Also see **"The WStringList Helper Interface" on page 272**.

CharToWString

Syntax

```
CharToWString(pczStr, iError, pczError)
```

Description

Translates the specified character array to a **WString** (wide string) object.

Parameters

Name	Type	Description
pczStr	const char*	Pointer to the character array.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

SBYN_WString*
Pointer to the **WString** object containing the translation of the character array.

DeleteWString

Syntax

```
DeleteWString(WStr, iError, pczError)
```

Description

Deletes the specified **WString** (wide string) object.

Parameters

Name	Type	Description
WStr	SBYN_WString*	Pointer to the WString object.

Return Value

None.

WStringToChar

Syntax

```
WStringToChar(WStr, iError, pczError)
```

Description

Translates the specified **WString** (wide string) object to a character array.

Parameters

Name	Type	Description
WStr	SBYN_WString*	Pointer to the WString object.

Return Value

char*
Pointer to the character array holding the translation.

8.1.22. The WStringList Helper Interface

The **WStringList** structure (wide string list—not part of the standard JMS API), is used to facilitate the handling of text strings.

The **WStringList** helper interface includes the following methods:

- **DeleteWStringList** on page 272
- **GetPropertyName** on page 272

Also see **"The WString Helper Interface" on page 270**.

DeleteWStringList

Syntax

```
DeleteWString(WStrList)
```

Description

Deletes the specified **WStringList** object (list of wide strings).

Parameters

Name	Type	Description
WStrList	SBYN_WString*	Pointer to the WStringList object.

Return Value

None.

GetPropertyName

Syntax

```
GetPropertyName(msg, iError, pczError)
```

Description

Retrieves a list of property names defined for the specified message.

Parameters

Name	Type	Description
msg	SBYN_Message*	Pointer to the message.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

SBYN_WStringList*
Pointer to the **WStringList** object (a list of wide strings) holding the property names.

8.1.23. Error Codes and Messages in the C API

The **C** API defines error codes as shown in Table 16.

Table 16 Error Codes Defined in the C API

Error Code	Explanation
0x80040300 = 2147746560 decimal JE_CODE_E_GENERAL	JMS exception, unspecified.
0x80040301 = 2147746561 decimal JE_CODE_E_REALLOC	A JMS exception occurred as a result of memory reallocation.
0x80040302 = 2147746562 decimal JE_CODE_E_MALLOC	A JMS exception occurred as a result of memory allocation.
0x80040303 = 2147746563 decimal JE_CODE_E_CONNECTION	A JMS exception occurred in setting up a connection.
0x80040304 = 2147746564 decimal JE_CODE_E_CREATION	A JMS exception occurred while creating a JMS object.
0x80040305 = 2147746565 decimal JE_CODE_E_CLOSED_SOCKET	A JMS exception occurred because of a closed socket.
0x80040306 = 2147746566 decimal JE_CODE_E_EOF	Processing ended because the BytesMessage or StreamMessage ended unexpectedly.
0x80040307 = 2147746567 decimal JE_CODE_E_NOTREADABLE	Processing ended because the message could not be read.
0x80040308 = 2147746568 decimal JE_CODE_E_NOTWRITEABLE	Processing ended because the message could not be written.
0x80040309 = 2147746569 decimal JE_CODE_E_FORMAT	Processing ended because the JMS client attempted to use a data type not supported by the message (or a message property), or attempted to read message data (or a message property) as the wrong type.
0x8004030A = 2147746570 decimal JE_CODE_E_ROLLBACK	The attempt to commit the session was unsuccessful of a transaction being rolled back.
0x8004030B = 2147746571 decimal JE_CODE_E_STATE	Processing ended because a method was invoked at an illegal or inappropriate time or because the provider was not in an appropriate state for the requested operation.
0x8004030C = 2147746572 decimal JE_CODE_E_DESTINATION	Processing ended because the destination could not be understood or was found to be invalid.
0x8004030D = 2147746573 decimal JE_CODE_E_NOTIMPL	Processing ended because a feature or interface was not implemented.
0x8004030E = 2147746574 decimal JE_CODE_E_INDEX_OUT_OF_BOUNDS	Processing ended because an index of some sort (such as to an array, to a string, or to a vector) was found to be outside the valid range.
0x8004030F = 2147746575 decimal JE_CODE_E_NULL_POINTER	Processing ended because the pointer in a case where an object was required.
0x80040310 = 2147746576 decimal JE_CODE_E_INVLD_CLIENT_ID	Processing ended because the connection's client ID was rejected by the provider.

Table 16 Error Codes Defined in the C API

Error Code	Explanation
0x80040311 = 2147746577 decimal JE_CODE_E_INVLD_SELECTOR	Processing ended because the message selector was found to be syntactically invalid.
0x80040312 = 2147746578 decimal JE_CODE_E_JMS_SECURITY	Processing was ended by JMS Security — for example, the provider rejected a name/password combination submitted by a client.
0x80040313 = 2147746579 decimal JE_CODE_E_RESOURCE_ALLOC	Processing ended because of the provider was unable to allocate resources required for the method/function.
0x80040314 = 2147746580 decimal JE_CODE_E_XA_IN_PROGRESS	Processing ended because a transaction was in progress.

8.2 Differences Between JMS Java API and SeeBeyond JMS C API

The **C** API has a few differences compared to the Java API for JMS, as noted below.

Differences Between Java and C in the BytesMessage Interface

For methods **ReadBytes()** and **WriteBytes()**, Java defines one signature to read or write the entire array (making use of Java's ability to easily determine the length of the given array), and another to read/write a portion of the array. Because **C** has no easy way to determine array length, the **C** API implements only the second functionality (reading or writing a portion of the array), and also provides a facility to see if more data exists.

Differences Between Java and C in the MapMessage Interface

The **C** API has not defined any equivalent for the Java **MapMessage** interface.

Differences Between Java and C in the MessageProducer Interface

The **C** API provides functions **GetJMS_ProducerID()** and **SetJMS_ProducerID**, which have no equivalent in the Java API.

Differences Between Java and C in Error Handling

Although C does not support exception-handling, the supplied wrappers for C use an error-handling scheme allows a JMS C++ exception to propagate to the application.

For each JMS C API, the final two parameters are an integer (*iErr*) and a text buffer (*szErrBuf*). Check the value of *iErr* after making any C API call; if the value is nonzero, it indicates that the underlying code has thrown an exception, and you should check the error description contained in the buffer.

For example:

```
pTopicConnection = CreateTopicConnection(pTcf, &iErr, szErrBuf);
if (iErr){
    printf("ERROR %d: %s\n", iErr, szErrBuf);
}
```

Chapter 9

Implementing JMS in C++

This chapter describes a sample implementation for the SeeBeyond Message Service using JMS in C++.

9.1 Implementing JMS Models in C++

This section discusses how to use the JMS C++ APIs to exchange data with an eGate system.

- **Session**—Characterized by a hostname, port number, session type (either a pub/sub "topic session" or a point-to-point "queue session"), and connection parameters such as acknowledgment mode, transaction mode, delivery mode, and client ID.
- **Destination**—Either a topic (for pub/sub messaging) or else a queue (for point-to-point messaging). Characterized by session and destination name.
- Message **Producer**—Either a topic publisher or a queue sender. Characterized by session and destination.
- Message **Consumer**—Either a topic subscriber or a queue receiver. Characterized by session and destination (and, in pub/sub messaging only, by the name of the durable subscriber, if designated).
- **Requestor**—In request/reply messaging, either a topic requestor or a queue requestor. Characterized by session, destination, message, and time-to-live value expressed in milliseconds.
- **Message**—Characterized by the message type and payload. The payload is also called the message *body*, or message *content*:
  - For messages of type BytesMessage, the payload is an array of bytes.

For messages of type TextMessage, the payload is a string of text characters. Native encoding is used.

9.2 Implementing Sample Code for Using JMS in C++

From the **Samples** directory located on the eGate installation CD-ROM, select the **C** folder and extract the sample files from this folder to the computer that you have installed the eGate API Kit on. The samples were built using Microsoft Visual Studio .Net 2003.

9.2.1. Creating Destinations

Destinations do not need to be created separately: they are created through the `session.createQueue()` and `session.createTopic()` functions. If these destinations do not exist, they are created automatically.

9.2.2. Using Mutlithreaded Apartments with C++

For the JMS API in C++, all the components were build using Microsoft Visual Studio .Net 2003 with the msvcp71.dll and the msvcr71.dll libraries. Applications that use the C++ components should be compiled with the runtime libraries option: "multi-threaded DLL" to assure the same runtime libraries are used.

For the JMS API in C++, the following sample programs are provided on the product CD-ROM:

- **Publish/Subscribe Messaging Using C++** on page 277
- **Queue Messaging (Sending/Receiving) Using C++** on page 281
- **Request-Reply Messaging Using C++** on page 285
- **Message Selector Using C++** on page 288
- **XA Publish/Subscribe Messaging For JMS Using C++** on page 292eGate

9.2.3. Publish/Subscribe Messaging Using C++

```
(1)    *-------------------------------------------------------------------
(2)    * Sample code to demonstrate JMS Pub/Sub.
(3)    *-------------------------------------------------------------------
(4)    *
(5)    *  Disclaimer:
(6)    *
(7)    *  Copyright 2002 by SeeBeyond Technology Corporation.
(8)    *  All Rights Reserved.
(9)    *   Unless otherwise stipulated in a written agreement for this
(10)   *  software, duly executed by SeeBeyond Technology Corporation, this
(11)   *  software is provided as is without warranty of any kind.  The
(12)   *  entire risk as to the results and performance of this software
(13)   *  is assumed by the user.SeeBeyond Technology Corporation disclaims
(14)   *  all warranties, either express or implied, including but not
(15)   *  limited, the implied warranties of merchantability, fitness for a
(16)   *  particular purpose, title and non-infringement, with respect to
(17)   *  this software.
(18)   *
(19)   * -------------------------------------------------------------------*/
(20)  #include    <ms.h>
```

```
(21)  #include    <mslocale.h>
(22)
(23)  #define                  MSCLIENT_DLL_NAME     "stc_msclient.dll"
(24)  char                    option1[] = "[ -u ] [ -p port ] [ -h hostname ]";
(25)  char                    option2[] = "[ -c ] [ -p port ] [ -h hostname ]";
(26)  char                    optdescription[] = "\t-u run as a
(27)                          producer\n\t-c run as a consumer\n\t-p port
(28)                          number\n\t-h       hostname\n";
(29)  static char             localHost[] = "localhost";
(30)  static unsigned short    susPort = 24053;       /* default port number */
(31)  unsigned long            sulMessageSize = 16;   /* default host name */
(32)  static char*             spHostName;
(33)  static void              subscriber();
(34)  static void              publisher();
(35)
(36)
(37)  #ifndef WIN32
(38)  #include <unistd.h>
(39)  #endif
(40)  #if defined(WIN32)
(41)  #include "sbyn_getopt.h"
(42)  #endif
(43)
(44)  #if defined(OS400)
(45)  extern char *optarg;
(46)  #endif
(47)
(48)  #if defined(__gnu__)
(49)  #include <getopt.h>
(50)  #endif
(51)
(52)
(53)  int main(int argc, char *argv[]) {
(54)      int           c;
(55)      char          cOption = 0;
(56)
(57)      spHostName = localHost;
(58)
(59)      while((c = getopt(argc, argv, ":p:h:P:H:cCUu")) != -1) {
(60)          switch(c){
(61)          case 'p':
(62)          case 'P':
(63)              susPort = atoi(optarg); /* setup the port number */
(64)              break;
(65)          case 'h':
(66)          case 'H':
(67)              spHostName = optarg; /* setup the hostname */
(68)              break;
(69)          case 'U':
(70)          case 'u':
(71)              cOption = 'u';/* run as a producer */
(72)              break;
(73)          case 'c':
(74)          case 'C':
(75)              cOption = 'c';/* run as a consumer */
(76)              break;
(77)          case ':':
(78)          case '?':
(79)              printf("\nSYNOPSIS\n");
(80)              printf("%s %s\n", argv[0], option1);
(81)              printf("%s %s\n", argv[0], option2);
(82)              printf("%s\n", optdescription);
(83)              exit(1);
(84)              break;
```

```
(85)          }
(86)      }
(87)
(88)      if (cOption == 'u'){
(89)          publisher();/* invoke producer */
(90)      } else if (cOption == 'c'){
(91)          subscriber();/* invoke consumer */
(92)      } else {
(93)          printf("\nSYNOPSIS\n");
(94)          printf("%s %s\n", argv[0], option1);
(95)          printf("%s %s\n", argv[0], option2);
(96)          printf("%s\n", optdescription);
(97)          exit(1);
(98)      }
(99)      return 0;
(100) }
(101)
(102)
(103) /*
(104)  * ============================================================
(105)  * Topic Publisher
(106)  * This routine demonstrates how to publish to a topic.
(107)  * ============================================================
(108) */
(109) static void publisher()
(110) {
(111)     char                    pBuffer[] = "This is a text message";
(112)     char                    pTopicName[] = "PubSubSample";
(113)     int                     iMessagePriority = 4;
(114)     long                    iTimeToLive = 0;
(115)
(116)     try {
(117)     /* Create a topic factory. */
(118)     TopicConnectionFactory* pTopicConnectionFactory =
(119)         LookupTopicConnectionFactory(MSCLIENT_DLL_NAME, "",
(120)         spHostName, susPort, 0, 0);
(121)
(122)     /* Create a topic connection. */
(123)     TopicConnection* pTopicConnection = pTopicConnectionFactory-
(124)     >createTopicConnection();
(125)
(126)     /* Set the client ID. */
(127)     pTopicConnection->setClientID("TopicPublisher");
(128)
(129)     /* Start the connection. */
(130)     pTopicConnection->start();
(131)
(132)     /* Create a topic session. */
(133)     TopicSession* pTopicSession = pTopicConnection-
(134)     >createTopicSession(true, Session::AUTO_ACKNOWLEDGE);
(135)
(136)     /* Create a topic. */
(137)     WString wsTopicName(pTopicName);
(138)     Topic* pTopic = pTopicSession->createTopic(wsTopicName);
(139)
(140)     /* Create a topic publisher. */
(141)     TopicPublisher* pTopicPublisher = pTopicSession-
(142)     >createPublisher(pTopic);
(143)
(144)     /* Create a text message. */
(145)     TextMessage* pTextMessage = pTopicSession->createTextMessage();
(146)
(147)     /* Clear the body (payload) of the message. */
(148)     pTextMessage->clearBody();
```

```
(149)
(150)     /* Copy in the text to be sent. */
(151)     pTextMessage->setText(pBuffer);
(152)
(153)     /* Set the JMSType of the message to "ASCII". */
(154)     pTextMessage->setJMSType("ASCII");
(155)
(156)     /* Publish the message. */
(157) cout << "Sending Message: " << pBuffer << endl;
(158)     pTopicPublisher->publish(pTextMessage);
(159)
(160)     /* Commit the session. */
(161)     pTopicSession->commit();
(162)
(163)     /* Close and clean up. */
(164)     pTopicPublisher->close();
(165)     pTopicSession->close();
(166)     pTopicConnection->close();
(167)     delete(pTextMessage);
(168)     delete(pTopicPublisher);
(169)     delete(pTopicSession);
(170)     delete(pTopicConnection);
(171)     delete(pTopicConnectionFactory);
(172)     }
(173)     catch (JMSException &e)
(174)     {
(175)        printf("JMS error: %s\n", e.getMessage());
(176)     }
(177) }
(178)
(179) /*
(180)  * ========================================================
(181)  * Topic Subscriber
(182)  * This routine demonstrates how to subsribe a message from
(183)  * a topic.
(184)  * ========================================================
(185) */
(186) static void subscriber() {
(187)     char                    szUserInput[80];
(188)     char                    pTopicName[] = "eGatePubSubSample";
(189)
(190)     try {
(191)     /* create a topic connection factory*/
(192)     TopicConnectionFactory*     pTopicConnectionFactory =
(193)         LookupTopicConnectionFactory(MSCLIENT_DLL_NAME, "", spHostName,
(194)         susPort, 0, 0);
(195)
(196)     /* create a topic connection */
(197)     TopicConnection* pTopicConnection = pTopicConnectionFactory-
(198)     >createTopicConnection();
(199)
(200)     /* set client ID */
(201)     pTopicConnection->setClientID("TopicSubscriber");
(202)
(203)     /* start connection */
(204)     pTopicConnection->start();
(205)
(206)     /* create a topic session */
(207)     TopicSession* pTopicSession = pTopicConnection-
(208)     >createTopicSession(true, Session::AUTO_ACKNOWLEDGE);
(209)
(210)
(211)     /* create a topic */
(212)     WString wsTopicName(pTopicName);
```

```
(213)     Topic* pTopic = pTopicSession->createTopic(wsTopicName);
(214)
(215)     /* create a subscriber */
(216)     TopicSubscriber* pTopicSubscriber = pTopicSession-
(217)     >createDurableSubscriber(pTopic, (char*)"TopicPublisher");
(218)
(219)     printf("Waiting for message ... \n");
(220)     TextMessage* pReceivedTextMessage;
(221)     do {
(222)         /* waiting for incoming messages */
(223)         Message* pReceivedMessage = pTopicSubscriber->receive();
(224)         pReceivedTextMessage = DYNAMIC_CAST(TextMessage,
(225)         pReceivedMessage);
(226) pTopicSession->commit();
(227)         if (pReceivedTextMessage){
(228)             WString wsJMSType = pReceivedTextMessage->getJMSType();
(229)             WString wsText = pReceivedTextMessage->getText();
(230)             string  strText;
(231)             strText = MsLocale::WideStringToString(wsText).c_str();
(232)             cout << "Received Text Message " << strText << endl;
(233)         }
(234)         printf("Enter 'r' for receiving more message, 'q' for exit\n");
(235)         scanf("%s", szUserInput);
(236)     } while ( szUserInput[0] != 'q' && szUserInput[0] != 'Q' );
(237)
(238)     /* close and delete objects */
(239)     pTopicSubscriber->close();
(240)     pTopicSession->close();
(241)     pTopicConnection->close();
(242)     delete(pReceivedTextMessage);
(243)     delete(pTopicSubscriber);
(244)     delete(pTopicSession);
(245)     delete(pTopicConnection);
(246)     delete(pTopicConnectionFactory);
(247)     }
(248)     catch (JMSException &e)
(249)     {
(250)          printf("JMS error: %s\n", e.getMessage());
(251)     }
(252) }
```

9.2.4. Queue Messaging (Sending/Receiving) Using C++

```
(1)   /* ---------------------------------------------------------------------
(2)    *   Sample code to demonstrate JMS Queue Messaging using C++.
(3)    *
(4)   * ---------------------------------------------------------------------
(5)    *
(6)    *  Disclaimer:
(7)    *
(8)    *   Copyright 2002 by SeeBeyond Technology Corporation.
(9)    *   All Rights Reserved.
(10)   *
(11)   *   Unless otherwise stipulated in a written agreement for this
(12)   *   software, duly executed by SeeBeyond Technology Corporation,
(13)   *   this software is provided as is without warranty of any kind.
(14)   *   The entire risk as to the results and performance of this software
(15)   *   is assumed by the user. SeeBeyond Technology Corporation disclaims
(16)   *   all warranties, either express or implied, including but not
(17)   *   limited, the implied warranties of merchantability, fitness for
(18)   *   a particular purpose, title and non-infringement, with respect
(19)   *   to this software.
(20)   * ---------------------------------------------------------------------*/
(21)  #include    <ms.h>
```

```
(22)  #include    <mslocale.h>
(23)
(24)  #ifndef WIN32
(25)  #include <unistd.h>
(26)  #endif
(27)  #if defined(WIN32)
(28)  #include "sbyn_getopt.h"
(29)  #endif
(30)
(31)  #if defined(OS400)
(32)  extern char *optarg;
(33)  #endif
(34)
(35)  #if defined(__gnu__)
(36)  #include <getopt.h>
(37)  #endif
(38)
(39)  #define   MSCLIENT_DLL_NAME     "stc_msclient.dll"
(40)
(41)  static    void sender();
(42)  static    void receiver();
(43)  char                            optionProducer[] = "[ -u ] [ -p port ]
(44)                                  [ -h hostname ]";
(45)  char                            optionConsumer[] = "[ -c ] [ -p port ]
(46)                                  [ -h hostname ]";
(47)  char                            optdescription[] = "\t-u run as a
(48)                                  producer\n\t-c run as a consumer\n\t-p
(49)                                  port number\n\t-h hostname\n";
(50)  char*                           spHostName;
(51)  char                            localHost[] = "localhost";
(52)  static unsigned short           susPort = 24053;
(53)  int                             iErr;
(54)  char                            szErrBuf[256];
(55)
(56)  int main(int argc, char *argv[]) {
(57)       int           c;
(58)       char          cOption = 0;
(59)
(60)       spHostName = localHost;
(61)
(62)       while((c = getopt(argc, argv, ":p:h:P:H:cCUu")) != -1) {
(63)            switch(c){
(64)            case 'p':
(65)            case 'P':
(66)                 susPort = atoi(optarg);  /* setup the port number */
(67)                 break;
(68)            case 'h':
(69)            case 'H':
(70)                 spHostName = optarg;     /* setup the hostname */
(71)                 break;
(72)            case 'U':
(73)            case 'u':
(74)                 cOption = 'u';           /* run as a producer */
(75)                 break;
(76)            case 'c':
(77)            case 'C':
(78)                 cOption = 'c';           /* run as a consumer */
(79)                 break;
(80)            case ':':
(81)            case '?':
(82)                 printf("\nSYNOPSIS\n");
(83)                 printf("%s %s\n", argv[0], optionProducer);
(84)                 printf("%s %s\n", argv[0], optionConsumer);
(85)                 printf("%s\n", optdescription);
```

```
(86)                exit(1);
(87)                break;
(88)           }
(89)      }
(90)
(91)      if (cOption == 'u'){
(92)           sender();             /* invoke producer */
(93)      } else if (cOption == 'c'){
(94)           receiver();           /* invoke consumer */
(95)      } else {
(96)           printf("\nSYNOPSIS\n");
(97)           printf("%s %s\n", argv[0], optionProducer);
(98)           printf("%s %s\n", argv[0], optionConsumer);
(99)           printf("%s\n", optdescription);
(100)          exit(1);
(101)     }
(102)  return 0;
(103) }
(104)
(105)
(106) /*
(107)  * ========================================================
(108)  * Queue Sender
(109)  * This routine demonstrates how to send message to a queue.
(110)  * ========================================================
(111) */
(112) static void sender()
(113) {
(114)     char                        pQueueName[] = "P2PSample";
(115)     const int                   MAX_MESSAGE_SIZE = 60;
(116)     char                        pBuffer[] = "This is a text message";
(117)
(118)     try {
(119)     /* Create a queue connection factory */
(120)     QueueConnectionFactory* pQueueConnectionFactory =
(121)          LookupQueueConnectionFactory(MSCLIENT_DLL_NAME, "",
      spHostName, susPort, 0, 0);
(122)
(123)     /* Create a queue connection */
(124)     QueueConnection* pQueueConnection = pQueueConnectionFactory-
      >createQueueConnection();
(125)
(126)     /* Set the client ID */
(127)     pQueueConnection->setClientID("SENDER");
(128)
(129)     /* Start the connection */
(130)     pQueueConnection->start();
(131)
(132)     /* Create a queue session */
(133)     QueueSession* pQueueSession = pQueueConnection-
      >createQueueSession(true, Session::CLIENT_ACKNOWLEDGE);
(134)
(135)     /* Create a queue */
(136)     WString wsQueueName(pQueueName);
(137)     Queue* pQueue = pQueueSession->createQueue(wsQueueName);
(138)
(139)     /* Create a queue sender */
(140)     QueueSender* pQueueSender = pQueueSession->createSender(pQueue);
(141)
(142)     /* Create a text message */
(143)     TextMessage* pTextMessage = pQueueSession->createTextMessage();
(144)
(145)     /* set the mode to r/w */
(146)     pTextMessage->clearBody();
```

```
(147)
(148)      /* Set the JMSType of the message to "ASCII". */
(149)      pTextMessage->setJMSType("ASCII");
(150)
(151)      /* Copy in the text to be sent. */
(152)      pTextMessage->setText(pBuffer);
(153)
(154)      /* send out the message */
(155)  cout << "Sending Text Message: " << pBuffer << endl;
(156)      pQueueSender->send(pTextMessage);
(157)
(158)      /* session commit */
(159)      pQueueSession->commit();
(160)
(161)
(162)      /* close and delete the objects */
(163)      pQueueSender->close();
(164)      pQueueSession->close();
(165)      pQueueConnection->close();
(166)      delete(pTextMessage);
(167)      delete(pQueueSender);
(168)      delete(pQueue);
(169)      delete(pQueueSession);
(170)      delete(pQueueConnection);
(171)      delete(pQueueConnectionFactory);
(172)      }
(173)      catch (JMSException &e)
(174)      {
(175)          printf("JMS error: %s\n", e.getMessage());
(176)      }
(177) }
(178)
(179) /*
(180)  * ============================================================
(181)  *  Queue Receiver
(182)  *  This routine demonstrates how to receive a message from a
(183)  *  queue.
(184)  * ============================================================
(185) */
(186) static void receiver() {
(187)      char                    szUserInput[80];
(188)      char                    pQueueName[] = "eGateP2PSample";
(189)
(190)      try {
(191)      /* Create a queue connection factory */
(192)      QueueConnectionFactory* pQueueConnectionFactory =
(193)           LookupQueueConnectionFactory(MSCLIENT_DLL_NAME, "",
      spHostName, susPort, 0, 0);
(194)
(195)      /* Create a queue connection */
(196)      QueueConnection* pQueueConnection = pQueueConnectionFactory-
      >createQueueConnection();
(197)
(198)      /* Set the client ID */
(199)      pQueueConnection->setClientID("RECEIVER");
(200)
(201)      /* Start the connection */
(202)      pQueueConnection->start();
(203)
(204)      /* Create a queue session */
(205)      QueueSession* pQueueSession = pQueueConnection-
      >createQueueSession(true, Session::CLIENT_ACKNOWLEDGE);
(206)
(207)      /* Create a queue */
```

```
(208)      WString wsQueueName(pQueueName);
(209)      Queue* pQueue = pQueueSession->createQueue(wsQueueName);
(210)
(211)      /* Create a queue receiver */
(212)      QueueReceiver* pQueueReceiver = pQueueSession-
      >createReceiver(pQueue);
(213)
(214)      printf("Waiting for message ... \n");
(215)      TextMessage* pReceivedTextMessage;
(216)      do {
(217)           /* waiting for incoming messages */
(218)           Message* pReceivedMessage = pQueueReceiver->receive();
(219)           pReceivedTextMessage = DYNAMIC_CAST(TextMessage,
      pReceivedMessage);
(220)           if (pReceivedTextMessage){
(221)                WString wsJMSType = pReceivedTextMessage->getJMSType();
(222)                WString wsText = pReceivedTextMessage->getText();
(223)                string  strText;
(224)                strText = MsLocale::WideStringToString(wsText).c_str();
(225)                cout << "Received Text Message " << strText << endl;
(226)           }
(227)           printf("Enter 'r' for receiving more message, 'q' for
      exit\n");
(228)           scanf("%s", szUserInput);
(229)      } while ( szUserInput[0] != 'q' && szUserInput[0] != 'Q' );
(230)
(231)      /* close and delete objects */
(232)      pQueueReceiver->close();
(233)      pQueueSession->close();
(234)      pQueueConnection->close();
(235)      delete(pReceivedTextMessage);
(236)      delete(pQueueReceiver);
(237)      delete(pQueueSession);
(238)      delete(pQueueConnection);
(239)      delete(pQueueConnectionFactory);
(240)      }
(241)      catch (JMSException &e)
(242)      {
(243)          printf("JMS error: %s\n", e.getMessage());
(244)      }
(245) }
```

9.2.5. Request-Reply Messaging Using C++

```
(1)   /* ------------------------------------------------------------------
(2)    * Sample code to demonstrate JMS Request-Reply messaging using C++
(3)   * ------------------------------------------------------------------
(4)   *
(5)   *  Disclaimer:
(6)   *
(7)   *   Copyright 2002 by SeeBeyond Technology Corporation.
(8)   *   All Rights Reserved.
(9)   *
(10)  *   Unless otherwise stipulated in a written agreement for this
(11)  *   software,
(12)  *   duly executed by SeeBeyond Technology Corporation, this software is
(13)  *   provided as is without warranty of any kind.  The entire risk as to
(14)  *   the results and performance of this software is assumed by the
(15)  *   user. SeeBeyond Technology Corporation disclaims all warranties,
(16)  *   either
(17)  *   express or implied, including but not limited, the implied
(18)  *   warranties
(19)  *   of merchantability, fitness for a particular purpose, title and
(20)  *   non-infringement, with respect to this software.
```

```
(21)  * --------------------------------------------------------------------*/
(22)
(23)  #include    <ms.h>
(24)  #include    <mslocale.h>
(25)
(26)  #ifndef WIN32
(27)  #include <unistd.h>
(28)  #endif
(29)  #if defined(WIN32)
(30)  #include "sbyn_getopt.h"
(31)  #endif
(32)
(33)  #if defined(OS400)
(34)  extern char *optarg;
(35)  #endif
(36)
(37)  #if defined(__gnu__)
(38)  #include <getopt.h>
(39)  #endif
(40)
(41)  #define                 MSCLIENT_DLL_NAME     "stc_msclient.dll"
(42)
(43)  static void requestor();
(44)
(45)  char                        optionRequestor[] = "[ -r ] [ -p port ]
(46)                              [ -h hostname ]";
(47)  char                        optdescription[] = "\t-r   run as a
(48)                              requestor\n\t-
(49)                              p port number\n\t-h  hostname\n";
(50)  char*                       spHostName;
(51)  char                        localHost[] = "localhost";
(52)  static unsigned short       susPort = 24053;
(53)  char                        pQueueName[] = "QueueRequestorSample";
(54)
(55)  int main(int argc, char *argv[]) {
(56)       int           c;
(57)       char          cOption = 0;
(58)
(59)       spHostName = localHost;
(60)
(61)       while((c = getopt(argc, argv, ":p:h:P:H:rR")) != -1) {
(62)            switch(c){
(63)            case 'p':
(64)            case 'P':
(65)                 susPort = atoi(optarg);  /* setup the port number */
(66)                 break;
(67)            case 'h':
(68)            case 'H':
(69)                 spHostName = optarg;     /* setup the hostname */
(70)                 break;
(71)            case 'R':
(72)            case 'r':
(73)                 cOption = 'r';           /* run as a requestor */
(74)                 break;
(75)            case ':':
(76)            case '?':
(77)                 printf("\nSYNOPSIS\n");
(78)                 printf("%s %s\n", argv[0], optionRequestor);
(79)                 printf("%s\n", optdescription);
(80)                 exit(1);
(81)             }
(82)       }
(83)
(84)       if (cOption == 'r'){
```

```
(85)            requestor();/* invoke requestor */
(86)       } else {
(87)            printf("\nSYNOPSIS\n");
(88)            printf("%s %s\n", argv[0], optionRequestor);
(89)            printf("%s\n", optdescription);
(90)            exit(1);
(91)       }
(92)   return 0;
(93)  }
(94)
(95)
(96)
(97)  /*
(98)   * ============================================================
(99)   * Queue Requestor
(100)  * This routine demonstrates how to do Request/Reply.
(101)  * ============================================================
(102) */
(103) void requestor(){
(104)      char pBuffer[] = "This is a text message";
(105)
(106)      try {
(107)      /* Create a queue connection factory */
(108)      QueueConnectionFactory* pQueueConnectionFactory =
(109)           LookupQueueConnectionFactory(MSCLIENT_DLL_NAME, "",
(110)           spHostName, susPort, 0, 0);
(111)
(112)      /* Create a queue connection */
(113)      QueueConnection* pQueueConnection = pQueueConnectionFactory-
(114)      >createQueueConnection();
(115)
(116)      /* Set the client ID */
(117)      pQueueConnection->setClientID("REQUESTOR");
(118)
(119)      /* Start the connection */
(120)      pQueueConnection->start();
(121)
(122)
(123)      /* Create a queue session */
(124)      QueueSession* pQueueSession = pQueueConnection-
(125)      >createQueueSession(false, Session::CLIENT_ACKNOWLEDGE);
(126)
(127)      /* Create a queue */
(128)      WString wsQueueName(pQueueName);
(129)      Queue* pQueue = pQueueSession->createQueue(wsQueueName);
(130)
(131)      /* Create a queue requestor */
(132)     QueueRequestor* pQueueRequestor = new QueueRequestor(pQueueSession,
(133)      pQueue);
(134)
(135)
(136)      /* Create a text message */
(137)      TextMessage* pTextMessage = pQueueSession->createTextMessage();
(138)
(139)      /* set the mode to r/w */
(140)      pTextMessage->clearBody();
(141)
(142)       /* Copy in the text to be sent. */
(143)      pTextMessage->setText(pBuffer);
(144)
(145)      /* Set ReplyTo destination */
(146)      pTextMessage->setJMSReplyTo(pQueue);
(147)
(148)      /* Make a request and wait for a reply */
```

```
(149)      Message* pReplyMessage = pQueueRequestor->request(pTextMessage,
(150)      100000);
(151)      TextMessage* pReplyTextMessage = DYNAMIC_CAST(TextMessage,
(152)      pReplyMessage);
(153)      if (pReplyTextMessage){
(154)        WString wsText = pReplyTextMessage->getText();
(155)        string  strText;
(156)        strText = MsLocale::WideStringToString(wsText).c_str();
(157)        cout << "Received Text Message " << strText << endl;
(158)      }
(159)      delete(pReplyTextMessage);
(160)
(161)       /* close and delete objects */
(162)      pQueueSession->close();
(163)      pQueueConnection->close();
(164)      delete(pTextMessage);
(165)      delete(pQueueRequestor);
(166)      delete(pQueue);
(167)      delete(pQueueSession);
(168)      delete(pQueueConnection);
(169)      delete(pQueueConnectionFactory);
(170)      }
(171)      catch (JMSException &e)
(172)      {
(173)          printf("JMS error: %s\n", e.getMessage());
(174)      }
(175) }
(176)
```

9.2.6. Message Selector Using C++

```
(1)    *------------------------------------------------------------------
(2)    * Sample code to demonstrate JMS pub/sub messaging with selector.
(3)    *------------------------------------------------------------------
(4)    *
(5)    *  Disclaimer:
(6)    *
(7)    *  Copyright 2002 by SeeBeyond Technology Corporation.
(8)    *  All Rights Reserved.
(9)    *   Unless otherwise stipulated in a written agreement for this
(10)   *  software, duly executed by SeeBeyond Technology Corporation, this
(11)   *  software is provided as is without warranty of any kind.  The
(12)   *  entire risk as to the results and performance of this software
(13)   *  is assumed by the user.SeeBeyond Technology Corporation disclaims
(14)   *  all warranties, either express or implied, including but not
(15)   *  limited, the implied warranties of merchantability, fitness for a
(16)   *  particular purpose, title and non-infringement, with respect to
(17)   *  this software.
(18)   *
(19)   * ------------------------------------------------------------------*/
(20)
(21)  #include    <ms.h>
(22)  #include    <mslocale.h>
(23)
(24)  #ifndef WIN32
(25)  #include <unistd.h>
(26)  #endif
(27)  #if defined(WIN32)
(28)  #include "sbyn_getopt.h"
(29)  #endif
(30)
(31)  #if defined(OS400)
(32)  extern char *optarg;
(33)  #endif
```

```
(34)
(35)  #if defined(__gnu__)
(36)  #include <getopt.h>
(37)  #endif
(38)
(39)  #define                  MSCLIENT_DLL_NAME     "stc_msclient.dll"
(40)  char                     optionProducer[] = "[ -u ] [ -p port ]
(41)                           [ -h hostname ]";
(42)  char                     optionConsumer[] = "[ -c ] [ -p port ]
(43)                           [ -h hostname ]";
(44)  char                     optdescription[] = "\t-u run as a
(45)                           producer\n\t-c run as a consumer\n\t-p
(46)                           port number\n\t-h hostname\n";
(47)  static char              localHost[] = "localhost";
(48)  static unsigned short    susPort = 24053; /* default port number */
(49)  unsigned long            sulMessageSize = 16; /* default host name */
(50)  static char*             spHostName;
(51)  static char PROP_NAME[] = "property";
(52)
(53)
(54)  static void selector_publisher();
(55)  static void selector_subscriber();
(56)
(57)
(58)  int main(int argc, char *argv[]) {
(59)       int           c;
(60)       char          cOption = 0;
(61)
(62)       spHostName = localHost;
(63)
(64)       while((c = getopt(argc, argv, ":p:h:P:H:cCUu")) != -1) {
(65)            switch(c){
(66)            case 'p':
(67)            case 'P':
(68)                 susPort = atoi(optarg);  /* setup the port number */
(69)                 break;
(70)            case 'h':
(71)            case 'H':
(72)                 spHostName = optarg;     /* setup the hostname */
(73)                 break;
(74)            case 'U':
(75)            case 'u':
(76)                 cOption = 'u';           /* run as a producer */
(77)                 break;
(78)            case 'c':
(79)            case 'C':
(80)                 cOption = 'c';           /* run as a consumer */
(81)                 break;
(82)            case ':':
(83)            case '?':
(84)                 printf("\nSYNOPSIS\n");
(85)                 printf("%s %s\n", argv[0], optionProducer);
(86)                 printf("%s %s\n", argv[0], optionConsumer);
(87)                 printf("%s\n", optdescription);
(88)                 exit(1);
(89)                 break;
(90)            }
(91)       }
(92)
(93)       if (cOption == 'u'){
(94)            selector_publisher();          /* invoke producer */
(95)       } else if (cOption == 'c'){
(96)            selector_subscriber();         /* invoke consumer */
(97)       } else {
```

```
(98)            printf("\nSYNOPSIS\n");
(99)            printf("%s %s\n", argv[0], optionProducer);
(100)           printf("%s %s\n", argv[0], optionConsumer);
(101)           printf("%s\n", optdescription);
(102)           exit(1);
(103)       }
(104)  return 0;
(105) }
(106)
(107) /*
(108)  * ============================================================
(109)  * Topic Publisher
(110)  * This routine demonstrates how to publish to a topic.
(111)  * ============================================================
(112) */
(113) static void selector_publisher(){
(114)       int                                 ii;
(115)       static char                         pTopicName[] = "Selector";
(116)
(117)       try {
(118)       /* Create a topic factory. */
(119)       TopicConnectionFactory* pTopicConnectionFactory =
(120)            LookupTopicConnectionFactory(MSCLIENT_DLL_NAME, "",
(121)            spHostName, susPort, 0, 0);
(122)
(123)       /* Create a topic connection. */
(124)       TopicConnection* pTopicConnection = pTopicConnectionFactory-
(125)       >createTopicConnection();
(126)
(127)       /* Set the client ID. */
(128)       pTopicConnection->setClientID("TopicPublisher");
(129)
(130)       /* Start the connection. */
(131)       pTopicConnection->start();
(132)
(133)       /* Create a topic session. */
(134)       TopicSession* pTopicSession = pTopicConnection-
(135)       >createTopicSession(true, Session::AUTO_ACKNOWLEDGE);
(136)
(137)       /* Create a topic. */
(138)       WString wsTopicName(pTopicName);
(139)       Topic* pTopic = pTopicSession->createTopic(wsTopicName);
(140)
(141)       /* Create a topic publisher. */
(142)       TopicPublisher* pTopicPublisher = pTopicSession-
(143)       >createPublisher(pTopic);
(144)
(145)
(146)       /* Set delivery mode as persistent */
(147)       pTopicPublisher->setDeliveryMode(DeliveryMode::PERSISTENT);
(148)
(149)
(150)       /* publish 10 messages to the topic */
(151)       TextMessage* msglist[10];
(152)       for(ii=0; ii<10 ;ii++){
(153)            int index;
(154)            char buf[80];
(155)
(156)            /* Create a text message. */
(157)            msglist[ii] = pTopicSession->createTextMessage();
(158)
(159)            /* Clear the body (payload) of the message. */
(160)            msglist[ii]->clearBody();
(161)            index = ii % 10;
```

```
(162)           sprintf(buf, "%d", index);
(163)
(164)           /* Set the string property */
(165)           msglist[ii]->setStringProperty(PROP_NAME, buf);
(166)
(167)           /* Copy in the text to be sent. */
(168)           msglist[ii]->setText("This is a text message");
(169)
(170)           /* Publish the message. */
(171)           pTopicPublisher->publish(msglist[ii]);
(172)           printf("... Published 1 message with property %s = %d\n",
(173)           PROP_NAME, ii);
(174)      }
(175)
(176)      /* Commit the session. */
(177)      pTopicSession->commit();
(178)
(179)
(180)      /* close and delete objects */
(181)      pTopicPublisher->close();
(182)      pTopicSession->close();
(183)      pTopicConnection->close();
(184)      for (ii = 0; ii < 10; ii++){
(185)           delete(msglist[ii]);
(186)      }
(187)      delete(pTopicPublisher);
(188)      delete(pTopicSession);
(189)      delete(pTopicConnection);
(190)      delete(pTopicConnectionFactory);
(191)      }
(192)      catch (JMSException &e)
(193)      {
(194)          printf("JMS error: %s\n", e.getMessage());
(195)      }
(196) }
(197)
(198) /*
(199)  * ============================================================
(200)  * Topic Subscriber
(201)  * This routine demonstrates how to subsribe a message from a
(202)  * topic.
(203)  * ============================================================
(204)  */
(205) static void selector_subscriber(){
(206)      char                 selectorString[80];
(207)      char                 selectorSubscriberName[80];
(208)      int                  selector = 7;
(209)      char*                selectorName;
(210)      char                 pTopicName[] = "eGateSelector";
(211)
(212)      try {
(213)      /* create a topic connection factory */
(214)      TopicConnectionFactory* pTopicConnectionFactory =
(215)           LookupTopicConnectionFactory(MSCLIENT_DLL_NAME, "",
(216)      spHostName, susPort, 0, 0);
(217)
(218)      /* Create a topic connection. */
(219)      TopicConnection* pTopicConnection = pTopicConnectionFactory-
(220)      >createTopicConnection();
(221)
(222)      /* Set the client ID. */
(223)      pTopicConnection->setClientID("Publisher");
(224)
(225)      /* Start the connection. */
```

```
(226)      pTopicConnection->start();
(227)
(228)      /* Create a topic session. */
(229)      TopicSession* pTopicSession = pTopicConnection-
      >createTopicSession(true, Session::AUTO_ACKNOWLEDGE);
(230)
(231)      /* Create a topic. */
(232)      WString wsTopicName(pTopicName);
(233)      Topic* pTopic = pTopicSession->createTopic(wsTopicName);
(234)
(235)
(236)      /* Create subscriber  with selector*/
(237)      sprintf(selectorString, "%s = '%d'", PROP_NAME, selector);
(238)      sprintf(selectorSubscriberName, "SelectorSubscriber%d", selector);
(239)      TopicSubscriber* pTopicSubscriber = pTopicSession-
(240)      >createDurableSubscriber(pTopic, selectorSubscriberName,
(241)      selectorString, 0);
(242)
(243)      /* Get message using selector */
(244)      selectorName = pTopicSubscriber->getMessageSelector();
(245)      printf("using selector: %s\n", selectorName);
(246)
(247)      Message* pMessage;
(248)      for (pMessage = pTopicSubscriber->receive();
(249)           pMessage != 0;
(250)           pMessage = pTopicSubscriber->receive(1000))
(251)      {
(252)           string strProperty =  MsLocale::WideStringToString(pMessage-
(253)           >getStringProperty(PROP_NAME)).c_str();
(254)           cout << "Received 1 message with " << PROP_NAME << " = " <<
(255)           strProperty << endl;;
(256)           delete(pMessage);
(257)      }
(258)
(259)      /* Session commit */
(260)      pTopicSession->commit();
(261)
(262)      /* Close and delete objects */
(263)      pTopicSubscriber->close();
(264)      pTopicSession->close();
(265)      pTopicConnection->close();
(266)      delete(pTopicSubscriber);
(267)      delete(pTopicSession);
(268)      delete(pTopicConnection);
(269)      delete(pTopicConnectionFactory);
(270)      }
(271)      catch (JMSException &e)
(272)      {
(273)          printf("JMS error: %s\n", e.getMessage());
(274)      }
(275) }
```

9.2.7. XA Publish/Subscribe Messaging For JMS Using C++

```
(1)    *-------------------------------------------------------------------
(2)    * Sample code to demonstrate JMS Pub/Sub in XA.
(3)    *-------------------------------------------------------------------
(4)    *
(5)    *  Disclaimer:
(6)    *
(7)    *  Copyright 2002 by SeeBeyond Technology Corporation.
(8)    *  All Rights Reserved.
(9)    *   Unless otherwise stipulated in a written agreement for this
(10)   *  software, duly executed by SeeBeyond Technology Corporation, this
```

```
(11)   *  software is provided as is without warranty of any kind.  The
(12)   *  entire risk as to the results and performance of this software
(13)   *  is assumed by the user.SeeBeyond Technology Corporation disclaims
(14)   *  all warranties, either express or implied, including but not
(15)   *  limited, the implied warranties of merchantability, fitness for a
(16)   *  particular purpose, title and non-infringement, with respect to
(17)   *  this software. *
(18)   * ---------------------------------------------------------------------
(19)  #include    <ms.h>
(20)  #include    <msxa.h>
(21)  #include    <mslocale.h>
(22)
(23)  #ifndef WIN32
(24)  #include <unistd.h>
(25)  #endif
(26)  #if defined(WIN32)
(27)  #include "sbyn_getopt.h"
(28)  #endif
(29)
(30)  #if defined(OS400)
(31)  extern char *optarg;
(32)  #endif
(33)
(34)  #if defined(__gnu__)
(35)  #include <getopt.h>
(36)  #endif
(37)
(38)  char                    optionProducer[] = "[ -u ] [ -p port ] [ -h
      hostname ]";
(39)  char                    optionConsumer[] = "[ -c ] [ -p port ] [ -h
      hostname ]";
(40)  char                    optdescription[] = "\t-u        run as a
      producer\n\t-c run as a consumer\n\t-p port number\n\t-h
      hostname\n";
(41)  static char             localHost[] = "localhost";
(42)  static unsigned short   susPort = 24053; /* default port number */
(43)  unsigned long           sulMessageSize = 16; /* default host name */
(44)  static char*            spHostName;
(45)  static int              iNumMessages = 10;
(46)  static char             szText[] = "This is a text message";
(47)
(48)  static void XATopicPub();
(49)  static void XATopicSub();
(50)
(51)  #define MSCLIENT_DLL_NAME "stc_msclient.dll"
(52)
(53)
(54)  int main(int argc, char *argv[]) {
(55)       int           c;
(56)       char          cOption = 0;
(57)
(58)       spHostName = localHost;
(59)
(60)       while((c = getopt(argc, argv, ":p:h:P:H:cCUu")) != -1) {
(61)            switch(c){
(62)            case 'p':
(63)            case 'P':
(64)                 susPort = atoi(optarg);  /* setup the port number */
(65)                 break;
(66)            case 'h':
(67)            case 'H':
(68)                 spHostName = optarg;     /* setup the hostname */
(69)                 break;
(70)            case 'U':
```

```
(71)            case 'u':
(72)                 cOption = 'u';           /* run as a producer */
(73)                 break;
(74)            case 'c':
(75)            case 'C':
(76)                 cOption = 'c';           /* run as a consumer */
(77)                 break;
(78)            case ':':
(79)            case '?':
(80)                 printf("\nSYNOPSIS\n");
(81)                 printf("%s %s\n", argv[0], optionProducer);
(82)                 printf("%s %s\n", argv[0], optionConsumer);
(83)                 printf("%s\n", optdescription);
(84)                 exit(1);
(85)                 break;
(86)            }
(87)       }
(88)
(89)       if (cOption == 'u'){
(90)            XATopicPub();          /* invoke producer */
(91)       } else if (cOption == 'c'){
(92)            XATopicSub();         /* invoke consumer */
(93)       } else {
(94)            printf("\nSYNOPSIS\n");
(95)            printf("%s %s\n", argv[0], optionProducer);
(96)            printf("%s %s\n", argv[0], optionConsumer);
(97)            printf("%s\n", optdescription);
(98)            exit(1);
(99)       }
(100)      return 0;
(101) }
(102)
(103) /*
(104)  * ============================================================
(105)  * Publish Message
(106)  *   This routine publishes iNumMessages to the topic
(107)  * ============================================================
(108) */
(109) static void PublishMessage(TopicPublisher* pPublisher, Message*
      pMessage, int iNumMessages)
(110) {
(111)      int ii;
(112)      for ( ii = 0; ii < iNumMessages; ii++){
(113)           pMessage->setIntProperty("Sequence", ii);
(114)           printf("Sending Message: Sequence number %d\n", ii);
(115)           pPublisher->publish(pMessage);
(116)      }
(117) }
(118)
(119)
(120) /*
(121)  * ============================================================
(122)  * Receive Message
(123)  *   This routine block on receiving message for maximum iWait
(124)  *   seconds before return.
(125)  * ============================================================
(126) */
(127) static int SubscriberReceiveMessage(TopicSubscriber* pSub,
      TopicSession* pSession)
(128) {
(129)      int iMsgCount = 0;
(130)      Message* pRMsg = 0;
(131)      char szUserInput[8];
(132)      printf("Waiting for message ... \n");
```

```
(133)      do {
(134)           pRMsg = pSub->receive();
(135)           printf("Received Message %d\n", iMsgCount);
(136)           iMsgCount++;
(137)           if (iMsgCount >= iNumMessages){
(138)                printf("Enter 'r' for receiving more message, 'q' for
      exit\n");
(139)                scanf("%s", szUserInput);
(140)                pSession->commit();
(141)                iMsgCount = 0;
(142)           }
(143)
(144)      } while(szUserInput[0] != 'q' && szUserInput[0] != 'Q');
(145)      return iMsgCount;
(146) }
(147)
(148) /*
(149)  * ============================================================
(150)  * Topic Publisher
(151)  *   This routine demonstrates how to publish to a topic.
(152)  * ============================================================
(153) */
(154) void XATopicPub()
(155) {
(156)      char                              pTopicName[] = "XAPubSubSample";
(157)      Xid *pXid;
(158)
(159)      try {
(160)      /* Create a topic factory. */
(161)      XATopicConnectionFactory* pXATopicConnectionFactory =
(162)           LookupXATopicConnectionFactory(MSCLIENT_DLL_NAME, "",
(163)      spHostName, susPort, 0, 0);
(164)
(165)      /* Create a topic connection. */
(166)      XATopicConnection* pXATopicConnection = pXATopicConnectionFactory-
(167)      >createXATopicConnection();
(168)
(169)      /* set client ID */
(170)      pXATopicConnection->setClientID("eGate{7E527692-770A-11D5-B139-
(171)      935EB6E85DBD}");
(172)
(173)      /* create XA session */
(174)      XATopicSession* pXATopicSession = pXATopicConnection-
(175)      >createXATopicSession();
(176)
(177)      /* get session */
(178)      TopicSession* pTopicSession = pXATopicSession->getTopicSession();
(179)
(180)      /* get XA resource */
(181)      XAResource* pXATopicResource = pXATopicSession->getXAResource();
(182)
(183)      /* create a Topic */
(184)      Topic* pTopic = pTopicSession->createTopic(pTopicName);
(185)
(186)      /* create a publisher */
(187)      TopicPublisher* pTopicPublisher = pTopicSession-
(188)      >createPublisher(pTopic);
(189)
(190)      /* connection start */
(191)      pXATopicConnection->start();
(192)
(193)      /* create xa id */
(194)      pXid = CreateXid(MSCLIENT_DLL_NAME);
(195)
```

```
(196)      /* associate the global transaction with the resource */
(197)      pXATopicResource->start(pXid, XAResource::TMNOFLAGS);
(198)
(199)      /* create a meesage */
(200)      TextMessage* pMessage = pXATopicSession->createTextMessage();
(201)
(202)      /* set mode to r/w */
(203)      pMessage->clearBody();
(204)
(205)      /* write bytes */
(206)      pMessage->setText((char*)szText);
(207)
(208)      /* publish message */
(209)      printf("Sending %d messages\n", iNumMessages);
(210)      pTopicPublisher->publish(pMessage);
(211)
(212)      /* xaEnd */
(213)      pXATopicResource->end(pXid, XAResource::TMSUCCESS);
(214)
(215)      /* ====================================================
(216)       * Prepare-Rollback
(217)       * ====================================================
(218)       */
(219)      /* xaPrepare */
(220)      pXATopicResource->prepare(pXid);
(221)
(222)      /* xaRollBack */
(223)      printf("Rolling back %d message\n", iNumMessages);
(224)      pXATopicResource->rollback(pXid);
(225)
(226)
(227)      /* ====================================================
(228)       * Prepare-Commit
(229)       * ====================================================
(230)       */
(231)
(232)      /* xa start */
(233)      pXATopicResource->start(pXid, XAResource::TMNOFLAGS);
(234)
(235)      /* send message */
(236)      printf("Sending %d messages\n", iNumMessages);
(237)      PublishMessage(pTopicPublisher, pMessage, iNumMessages);
(238)
(239)      /* xaEnd */
(240)      pXATopicResource->end(pXid, XAResource::TMSUCCESS);
(241)
(242)      /* xaPrepare */
(243)      if (XAResource::XA_OK != pXATopicResource->prepare(pXid))
(244)      {
(245)           printf("ERROR: XAResourcePrepare failed\n");
(246)      }
(247)
(248)      /* xa commit */
(249)      printf("Resource Commit...\n");
(250)      pXATopicResource->commit(pXid, true);
(251)
(252)      /* Close and clean up. */
(253)      pTopicPublisher->close();
(254)      pXATopicSession->close();
(255)      pXATopicConnection->close();
(256)      delete(pMessage);
(257)      delete(pTopicPublisher);
(258)      delete(pXid);
(259)      delete(pXATopicSession);
```

```
(260)      delete(pTopic);
(261)      delete(pXATopicConnection);
(262)      delete(pXATopicConnectionFactory);
(263)      }
(264)      catch (JMSException &e)
(265)      {
(266)          printf("JMS error: %s\n", e.getMessage());
(267)      }
(268) }
(269)
(270) /*
(271)  * ========================================================
(272)  *      Topic Subscriber
(273)  *  This routine demonstrates how to subsribe a message from a
(274)  *  topic.
(275)  * ========================================================
(276) */
(277) void XATopicSub()
(278) {
(279)      char                          pTopicName[] = "eGateXAPubSubSample";
(280)      int                             iNumReceived = 0;
(281)      char                            szUserInput[8];
(282)
(283)      try {
(284)      /* Create a topic factory. */
(285)      XATopicConnectionFactory* pXATopicConnectionFactory =
(286)           LookupXATopicConnectionFactory(MSCLIENT_DLL_NAME, "",
(287)      spHostName, susPort, 0, 0);
(288)
(289)      /* Create a topic connection. */
(290)      XATopicConnection* pXATopicConnection = pXATopicConnectionFactory-
(291)      >createXATopicConnection();
(292)
(293)
(294)      /* set client ID */
(295)      pXATopicConnection->setClientID("eGate{7E527692-770A-11D5-B139-
(296)      3456789}");
(297)
(298)      /* create XA session */
(299)      XATopicSession* pXATopicSession = pXATopicConnection-
(300)      >createXATopicSession();
(301)
(302)      /* get session */
(303)      TopicSession* pTopicSession = pXATopicSession->getTopicSession();
(304)
(305)      /* get XA resource */
(306)      XAResource* pXATopicResource = pXATopicSession->getXAResource();
(307)
(308)      /* create a Topic */
(309)      Topic* pTopic = pTopicSession->createTopic(pTopicName);
(310)
(311)      /* create a subscriber */
(312)      TopicSubscriber* pTopicSubscriber = pTopicSession-
(313)      >createDurableSubscriber(pTopic, (char*)"XATopicSubscriber");
(314)
(315)      /* connection start */
(316)      pXATopicConnection->start();
(317)
(318)      /* start xa resource */
(319)      Xid* pXid = CreateXid(MSCLIENT_DLL_NAME);
(320)
(321)
(322)      int iMsgCount = 0;
(323)      do {
```

```
(324)           Message* pRMsg = 0;
(325)           /* associate the global transaction with the resource */
(326)           pXATopicResource->start(pXid, XAResource::TMNOFLAGS);
(327)           while(iMsgCount < iNumMessages){
(328)                pRMsg = pTopicSubscriber->receive();
(329)                printf("Received Message %d\n", iMsgCount);
(330)                iMsgCount++;
(331)           }
(332)           /* xaEnd */
(333)           pXATopicResource->end(pXid, XAResource::TMSUCCESS);
(334)           pXATopicResource->commit(pXid, true);
(335)           printf("Enter 'r' for receiving more message, 'q' for
(336)           exit\n");
(337)           scanf("%s", szUserInput);
(338)
(339)           //pTopicSession->commit();
(340)           iMsgCount = 0;
(341)      } while(szUserInput[0] != 'q' && szUserInput[0] != 'Q');
(342)
(343)
(344)
(345)      /* close and delete objects */
(346)      pTopicSubscriber->close();
(347)      //pTopicSession->unsubscribe((char*)"TopicSubscriber");
(348)      pXATopicSession->close();
(349)      pXATopicConnection->close();
(350)
(351)      /* delete objects */
(352)      delete(pTopic);
(353)      delete(pTopicSubscriber);
(354)      delete(pXid);
(355)      delete(pXATopicSession);
(356)      delete(pXATopicConnection);
(357)      delete(pXATopicConnectionFactory);
(358)      }
(359)      catch (JMSException &e)
(360)      {
(361)          printf("JMS error: %s\n", e.getMessage());
(362)      }

       }
```

Chapter 10

Client Libraries for the Message Service C++ API

This chapter provides a detailed discussion of the Application Program Interfaces (APIs) for Message Service.

For C++

The SeeBeyond Message Service includes additional API support in the following languages:

For Java

For COM+

For C

10.1 The C++ API for the SeeBeyond Message Service

10.1.1. The Message Interface for JMS in C++

The `Message` interface is the root interface of all JMS messages. It defines the message header and the `acknowledge` method used for all messages.

Most message-oriented middle ware (MOM) products treat messages as lightweight entities that consist of a header and a payload. The header contains fields used for message routing and identification; the payload contains the application data being sent.

Within this general form, the definition of a message varies significantly across products. It would be quite difficult for the JMS API to support all of these message models.

With this in mind, the JMS message model has the following goals:

- Provide a single, unified message API
- Provide an API suitable for creating messages that match the format used by provider-native messaging applications
- Support the development of heterogeneous applications that span operating systems, machine architectures, and computer languages
- Support messages containing objects in the Java programming language ("Java objects")
- Support messages containing Extensible Markup Language (XML) pages

The Message methods are:

acknowledge on page 301
clearBody on page 301
clearBody on page 301
propertyExists on page 302
getBooleanProperty on page 302
getByteProperty on page 302
getDoubleProperty on page 303
getFloatProperty on page 303
getIntProperty on page 304
getLongProperty on page 304
getPropertyName on page 304
getShortProperty on page 305
getStringProperty on page 305
setBooleanProperty on page 306
setByteProperty on page 306
setDoubleProperty on page 306
setFloatProperty on page 307
setIntProperty on page 307
setLongProperty on page 308
setObjectProperty on page 308
setShortProperty on page 308
setStringProperty on page 309
propertyExists on page 309
getBooleanProperty on page 302
getByteProperty on page 302
getDoubleProperty on page 303
getFloatProperty on page 303
getIntProperty on page 304
getLongProperty on page 304
getObjectProperty on page 312
getShortProperty on page 305
getStringProperty on page 305
setBooleanProperty on page 313
setByteProperty on page 306
setDoubleProperty on page 306
setFloatProperty on page 307
setIntProperty on page 307
setLongProperty on page 308
setObjectProperty on page 308
setShortProperty on page 308
setStringProperty on page 309
getJMSCorrelationID on page 317
getJMSCorrelationIDAsBytes on page 317
getJMSDeliveryMode on page 317
getJMSExpiration on page 318
getJMSMessageID on page 318
setJMSPriority on page 321
getJMSRedelivered on page 318
getJMSReplyTo on page 319
getJMSTimestamp on page 319

getJMSType on page 319
getJMSCorrelationIDAsBytes on page 317
setJMSCorrelationID on page 319
setJMSDeliveryMode on page 320
setJMSExpiration on page 321
setJMSMessageID on page 321
setJMSPriority on page 321
setJMSRedelivered on page 322
setJMSReplyTo on page 322
setJMSTimestamp on page 322
setJMSType on page 323
getJMSCorrelationIDAsBytes on page 317
setJMSMessageID on page 324
setJMSType on page 323

acknowledge

Syntax

```
void acknowledge()
```

Description

Acknowledges the receipt of current and previous messages.

Return Value

None.

clearBody

Syntax

```
void clearBody()
```

Description

Clears the body of a message, leaving the message header values and property entries intact.

Return Value

None.

clearProperties

Syntax

```
void clearProperties()
```

Description

Clears the properties from a message, leaving the message header fields and body intact.

Return Value

None.

propertyExists

Syntax

```
STCBOOL propertyExists(name)
```

Description

Checks whether a value for a specific property exists.

Parameters

Name	Type	Description
name	WString&	The name of the property to check.

Return Value

STCBOOL

Returns **true** if the property value is defined; otherwise, returns **false**.

getBooleanProperty

Syntax

```
STCBOOL getBooleanProperty(name)
```

Description

Checks whether a value for a specific property exists.

Parameters

Name	Type	Description
name	WString&	The name of the property to check.

Return Value

STCBOOL

Returns **true** if the property value is defined; otherwise, returns **false**.

getByteProperty

Syntax

```
unsigned getByteProperty(name)
```

Description

Reads the value of the specified byte property in the specified message.

Parameters

Name	Type	Description
named	WString&	The name of the property to check.

Return Value

unsigned char

The value of the property.

getDoubleProperty

Syntax

```
double getDoubleProperty(name)
```

Description

Reads the value of the specified double (numeric) property in the specified message.

Parameters

Name	Type	Description
name	WString&	The name of the property to check.

Return Value

Double

The value of the property.

getFloatProperty

Syntax

```
float getFloatProperty(name)
```

Description

Reads the value of the specified floating-point (numeric) property in the specified message.

Parameters

Name	Type	Description
name	WString&	The name of the property to check.

Return Value

float

The value of the property.

getIntProperty

Syntax

```
int getIntProperty(name)
```

Description

Reads the value of the specified integer (numeric) property in the specified message.

Parameters

Name	Type	Description
name	WString&	The name of the property to check.

Return Value

int

The value of the property.

getLongProperty

Syntax

```
int64_t getLongProperty(name)
```

Description

Reads the value of the specified long (numeric) property in the specified message.

Parameters

Name	Type	Description
name	WString&	The name of the property to check.

Return Value

int64_t

The long value of the property.

getPropertyName

Syntax

```
wNamesList getPropertyName(name)
```

Description

Returns a list of WStrings for the specified message.

Parameters

Name	Type	Description
name	wNamesList	Returns a list of the property names.

Return Value

wNamesList

Returns a list.

getShortProperty

Syntax

```
short getShortProperty(name)
```

Description

Reads the value of the specified short (numeric) property in the specified message.

Parameters

Name	Type	Description
name	WString&	The name of the property to check.

Return Value

short

The value of the property.

getStringProperty

Syntax

```
WString getStringProperty(name)
```

Description

Reads the value of the specified text property in the specified message.

Parameters

Name	Type	Description
name	WString	The name of the property to check.

Return Value

WString

A WString (wide string) object containing the value of the property.

setBooleanProperty

Syntax

```
void setBooleanProperty(name, value)
```

Description

Writes a value for the specified Boolean property in the specified message.

Parameters

Name	Type	Description
name	WString&	The name of the property to check.
value	STCBool	The value to be written.

Return Value

None

setByteProperty

Syntax

```
void setByteProperty(name, value)
```

Description

Writes a value for the specified byte property in the specified message.

Parameters

Name	Type	Description
name	WString&	The name of the property to check.
value	unsigned char	The value to be written.

Return Value

None

setDoubleProperty

Syntax

```
void setDoubleProperty(name, value)
```

Description

Writes a value for the specified double (numeric) property in the specified message.

Parameters

Name	Type	Description
name	WString&	The name of the property to check.
value	double	The value to be written.

Return Value

None

setFloatProperty

Syntax

```
void setFloatProperty(name, value)
```

Description

Writes a value for the specified byte property in the specified message.

Parameters

Name	Type	Description
name	WString&	The name of the property to check.
value	float	The value to be written.

Return Value

None

setIntProperty

Syntax

```
void setIntProperty(name, value)
```

Description

Writes a value for the specified byte property in the specified message.

Parameters

Name	Type	Description
name	WString&	The name of the property to check.
value	int	The value to be written.

Return Value

None

setLongProperty

Syntax

```
void setLongProperty(name, value)
```

Description

Writes a value for the specified long (numeric) property in the specified message.

Parameters

Name	Type	Description
name	WString&	The name of the property to check.
User defined	long	The value to be written.

Return Value

None

setObjectProperty

Syntax

```
void setObjectProperty(name, value)
```

Description

Writes a value for the specified object property in the specified message.

Parameters

Name	Type	Description
name	WString&	The name of the property to check.
User defined	SerialObject	The value to be written.

Return Value

None

setShortProperty

Syntax

```
void setShortProperty(name, value)
```

Description

Writes a value for the specified short (numeric) property in the specified message.

Parameters

Name	Type	Description
name	WString&	The name of the property to check.
value	short	The value to be written.

Return Value

None

setStringProperty

Syntax

```
void setStringProperty(name, value)
```

Description

Writes a value for the specified text property in the specified message.

Parameters

Name	Type	Description
name	WString&	The name of the property to check.
value	char *	The value to be written.

Return Value

None

propertyExists

Syntax

```
STCBOOL propertyExists(name)
```

Description

Checks whether a value for a specific property exists.

Parameters

Name	Type	Description
name	char*	Pointer to the name of the property to check.

Return Value

STCBOOL

Returns **true** if the property value is defined; otherwise, returns **false**.

getBooleanProperty

Syntax

```
STCBOOL getBooleanProperty(name)
```

Description

Reads the value of the specified Boolean property in the specified message.

Parameters

Name	Type	Description
name	char*	Pointer to the name of the property to check.

Return Value

STCBOOL
The value of the property.

getByteProperty

Syntax

```
unsigned char getByteProperty(name)
```

Description

Reads the value of the specified byte property in the specified message.

Parameters

Name	Type	Description
name	char*	Pointer to the name of the property to check.

Return Value

unsigned char
The value of the property.

getDoubleProperty

Syntax

```
double getDoubleProperty(name)
```

Description

Reads the value of the specified double (numeric) property in the specified message.

Parameters

Name	Type	Description
name	char*	Pointer to the name of the property to check.

Return Value

double
The value of the property.

getFloatProperty

Syntax

```
float getFloatProperty(name)
```

Description

Reads the value of the specified floating-point (numeric) property in the specified message.

Parameters

Name	Type	Description
name	char*	Pointer to the name of the property to check.

Return Value

float
The value of the property.

getIntProperty

Syntax

```
int getIntProperty(name)
```

Description

Reads the value of the specified integer (numeric) property in the specified message.

Parameters

Name	Type	Description
name	char*	Pointer to the name of the property to check.

Return Value

int
The value of the property.

getLongProperty

Syntax

```
int64_t getLongProperty(name)
```

Description

Reads the value of the specified long (numeric) property in the specified message.

Parameters

Name	Type	Description
name	char*	Pointer to the name of the property to check.

Return Value

int64_t
The long value of the property.

getObjectProperty

Syntax

```
SerialObject getObjectProperty(name)
```

Description

Reads the value of the specified object property in the specified message.

Parameters

Name	Type	Description
name	char*	Pointer to the name of the object property to check.

Return Value

SerialObject
The serialized value of the property.

getShortProperty

Syntax

```
short getShortProperty(name)
```

Description

Reads the value of the specified short (numeric) property in the specified message.

Parameters

Name	Type	Description
name	char*	Pointer to the name of the property to check.

Return Value

short
The value of the property.

getStringProperty

Syntax

```
WString getStringProperty(name)
```

Description

Reads the value of the specified text property in the specified message.

Parameters

Name	Type	Description
name	char*	Pointer to the name of the property to check.

Return Value

WString*
Pointer to a **WString** (wide string) object containing the value of the property.

setBooleanProperty

Syntax

```
void setBooleanProperty(name, value)
```

Description

Writes a value for the specified Boolean property in the specified message.

Parameters

Name	Type	Description
name	char*	Pointer to the property name.
value	STCBOOL	The value to be written.

Return Value

None.

setByteProperty

Syntax

```
void setByteProperty(name, value)
```

Description

Writes a value for the specified byte property in the specified message.

Parameters

Name	Type	Description
name	char*	Pointer to the property name.
value	unsigned char	The value to be written.

Return Value

None.

setDoubleProperty

Syntax

```
void setDoubleProperty(name, value)
```

Description

Writes a value for the specified double (numeric) property in the specified message.

Parameters

Name	Type	Description
name	char*	Pointer to the property name.
value	double	The value to be written.

Return Value

None.

setFloatProperty

Syntax

```
void setFloatProperty(name, value)
```

Description

Writes a value for the specified floating-point (numeric) property in the specified message.

Parameters

Name	Type	Description
name	char*	Pointer to the property name.
Value	float	The value to be written.

Return Value

None.

setIntProperty

Syntax

```
void setIntProperty(name, value)
```

Description

Writes a value for the specified integer (numeric) property in the specified message.

Parameters

Name	Type	Description
name	char*	Pointer to the property name.
value	int	The value to be written.

Return Value

None.

setLongProperty

Syntax

```
void setLongProperty(name, value)
```

Description

Writes a value for the specified long (numeric) property in the specified message.

Parameters

Name	Type	Description
name	char*	Pointer to the property name.
value	int64_t	The value to be written.

Return Value

None.

setObjectProperty

Syntax

```
void setObjectProperty(name, value)
```

Description

Writes a value for the specified long (numeric) property in the specified message.

Parameters

Name	Type	Description
name	char*	Pointer to the property name.
value	SerialObject	The value to be written.

Return Value

None.

setShortProperty

Syntax

```
void setShortProperty(name, value)
```

Description

Writes a value for the specified short (numeric) property in the specified message.

Parameters

Name	Type	Description
name	char*	Pointer to the property name.
value	short	The value to be written.

Return Value

None.

setStringProperty

Syntax

```
void setStringProperty(name, value)
```

Description

Writes a value for the specified text property in the specified message.

Parameters

Name	Type	Description
name	char*	Pointer to the property name.
value	char*	The value to be written.

Return Value

None.

getJMSCorrelationID

Syntax

```
WString getJMSCorrelationID()
```

Description

Retrieves the correlation ID for the specified message.

Return Value

WString*
Pointer to a **WString** (wide string) object containing the text.

getJMSCorrelationIDAsBytes

Syntax

```
char* getJMSCorrelationIDAsBytes()
```

Description

Retrieves the correlation ID for the specified message as an array of characters.

Return Value

None

getJMSDeliveryMode

Syntax

```
int getJMSDeliveryMode()
```

Description

Retrieves the value of the **DeliveryMode** property for the specified message.

Return Value

None

getJMSExpiration

Syntax

```
int64_t getJMSExpiration()
```

Description

Retrieves the value of the timestamp set for the expiration of the specified message.

Return Value

int64_t
Long timestamp.

getJMSMessageID

Syntax

```
WString getJMSMessageID()
```

Description

Retrieves the message ID of the specified message.

Return Value

WString*
Pointer to a **WString** (wide string) object containing the text.

getJMSPriority

Syntax

```
int getJMSPriority()
```

Description

Retrieves the priority level for the specified message.

Return Value

int
The priority level.

getJMSRedelivered

Syntax

```
STCBOOL getJMSRedelivered()
```

Description

Retrieves an indication of whether the specified message is being redelivered.

Return Value

STCBOOL
Returns **true** if the message is being redelivered; otherwise, returns **false**.

getJMSReplyTo

Syntax

```
Destination* getJMSReplyTo()
```

Description

Retrieves the **Destination** object where a reply to the specified message should be sent (for request/reply messaging).

Return Value

SBYN_Destination*
Pointer to the **Destination** object.

getJMSTimestamp

Syntax

```
int64_t getJMSTimestamp()
```

Description

Retrieves the timestamp of the specified message.

Return Value

int64_t
Long timestamp.

getJMSType

Syntax

```
WString getJMSType()
```

Description

Gets the message type identifier supplied by the client when the message was sent.

Return Value

WString*
Pointer to a **WString** (wide string) object containing the text.

setJMSCorrelationID

Syntax

```
void setJMSCorrelationID(correlationID)
```

Description

Sets the correlation ID for the specified message.

Parameters

Name	Type	Description
correlationID	WString	The text string containing the correlation ID.

Return Value

None.

setJMSCorrelationIDAsBytes

Syntax

```
void setJMSCorrelationIDAsBytes(deliveryMode)
```

Description

Sets the correlation ID for the specified message.

Parameters

Name	Type	Description
deliveryMode	char *	Pointer to the character array containing the bytes for the correlation ID.

Return Value

None.

setJMSDeliveryMode

Syntax

```
void setJMSDeliveryMode(destination)
```

Description

Sets the delivery mode for the specified message.

Parameters

Name	Type	Description
destination	Destination*	Pointer to the value corresponding to the delivery mode.

Return Value

None.

setJMSExpiration

Syntax

```
void setJMSExpiration(expiration)
```

Description

Sets the timestamp at which the specified message is due to expire.

Parameters

Name	Type	Description
expiration	int64_t	Timestamp of the expiration.

Return Value

None.

setJMSMessageID

Syntax

```
void setJMSMessageID(id)
```

Description

Sets the message ID of the specified message.

Parameters

Name	Type	Description
id	WString	The text string containing the message ID.

Return Value

None.

setJMSPriority

Syntax

```
void setJMSPriority(priority)
```

Description

Sets the priority level for the specified message.

Parameters

Name	Type	Description
priority	int	The priority level.

Return Value

None.

setJMSRedelivered

Syntax

```
void setJMSRedelivered(redelivered)
```

Description

Determines whether to flag the specified message as being redelivered. Used, for example, to specify re-delivery for a message that has been sent but not acknowledged.

Parameters

Name	Type	Description
redelivered	STCBOOL	Flag: If **true**, the message is being redelivered.

Return Value

None.

setJMSReplyTo

Syntax

```
void setJMSReplyTo(replyTo)
```

Description

Sets the **Destination** object where a reply to the specified message should be sent.

Parameters

Name	Type	Description
replyto	Destination*	Pointer to the **Destination** object.

Return Value

None.

setJMSTimestamp

Syntax

```
void setJMSTimestamp(timestamp)
```

Description

Sets the timestamp (JMSTimestamp header field) that the specified message was handed off to a provider to be sent. Note that this is not necessarily the time the

message is actually transmitted; the actual **send** can occur later because of transactions or other client-side queueing of messages.

Parameters

Name	Type	Description
timestamp	int64_t	Timestamp to be set.

Return Value

None.

setJMSType

Syntax

```
void setJMSType(type)
```

Description

Sets the JMSType header field, which is often used to identify the message structure and the payload type.

Parameters

Name	Type	Description
type	WString	The text string containing the data.

Return Value

None.

setJMSCorrelationIDAsBytes

Syntax

```
void setJMSCorrelationIDAsBytes(correlationID)
```

Description

Sets the correlation ID for the specified message.

Parameters

Name	Type	Description
correlationID	char*	Pointer to the character array containing the bytes for the correlation ID.

Return Value

None.

setJMSMessageID

Syntax

```
void setJMSMessageID(id)
```

Description

Sets the message ID of the specified message.

Parameters

Name	Type	Description
id	char*	Pointer to the text string containing the message ID.

Return Value

None.

setJMSType

Syntax

```
void setJMSType(type)
```

Description

Sets the JMSType header field, which is often used to identify the message structure and the payload type.

Parameters

Name	Type	Description
pczJMSType	char*	Pointer to the text string containing the data.

Return Value

None.

10.1.2. The BytesMessage Interface for JMS in C++

A **BytesMessage** object is used for messages containing a stream of uninterpreted bytes. The receiver of the message supplies the interpretation of the bytes.

The BytesMessage methods are:

readBoolean on page 325
readByte on page 325
readChar on page 325
readDouble on page 326
readFloat on page 326
readInt on page 326
readLong on page 326

readBoolean

Syntax

```
STCBOOL readBoolean()
```

Description

Reads a Boolean value from the **BytesMessage** stream.

Return Value

STCBOOL
The value read from the **BytesMessage** stream.

readByte

Syntax

```
unsigned char readByte()
```

Description

Reads a single unsigned character from the **BytesMessage** stream.

Return Value

unsigned char
The value read from the **BytesMessage** stream.

readChar

Syntax

```
unsigned short readChar()
```

Description

Reads a single Unicode character from the **BytesMessage** stream.

Return Value

unsigned short
The value read from the **BytesMessage** stream.

readDouble

Syntax

```
double readDouble()
```

Description

Reads a double numeric value from the **BytesMessage** stream.

Return Value

double
The value read from the **BytesMessage** stream.

readFloat

Syntax

```
float readFloat()
```

Description

Reads a floating-point numeric value from the **BytesMessage** stream.

Return Value

float
The value read from the **BytesMessage** stream.

readInt

Syntax

```
int readInt()
```

Description

Reads a signed integer value from the **BytesMessage** stream.

Return Value

int
The value read from the **BytesMessage** stream.

readLong

Syntax

```
int64_t readLong(pMsg, iError, pczError)
```

Description

Reads a signed long integer from the **BytesMessage** stream.

Return Value

long
The value read from the **BytesMessage** stream.

---

readShort

Syntax

```
short readShort(pMsg, iError, pczError)
```

Description

Reads a signed short integer from the **BytesMessage** stream.

Return Value

short
The value read from the **BytesMessage** stream.

---

readUnsignedByte

Syntax

```
readUnsignedByte()
```

Description

Reads an unsigned short integer from the **BytesMessage** stream.

Return Value

int
The value read from the **BytesMessage** stream.

---

readUnsignedShort

Syntax

```
int readUnsignedShort()
```

Description

Reads an unsigned short integer from the **BytesMessage** stream.

Return Value

int
The value read from the **BytesMessage** stream.

---

readUTF

Syntax

```
WString readUTF()
```

Description

Reads the value of a string that has been encoded using a modified UTF-8 format from the **BytesMessage** stream.

Return Value

WString*
Pointer to a **WString** (wide string) object containing the text from the **BytesMessage** stream.

reset

Syntax

```
void reset()
```

Description

Puts the message body into read-only mode and repositions the stream of bytes to the beginning.

Return Value

None.

writeBoolean

Syntax

```
void writeBoolean(value)
```

Description

Writes a Boolean value to the **BytesMessage** stream.

Parameters

Name	Type	Description
value	STCBOOL	The value to be written.

Return Value

None.

writeByte

Syntax

```
void writeByte(value)
```

Description

Writes a single byte (unsigned char) to the **BytesMessage** stream.

Parameters

Name	Type	Description
value	unsigned char	The value to be written.

Return Value

None.

writeBytes

Syntax

```
void writeBytes(value)
```

Description

Writes an array of bytes (unsigned char values) to the **BytesMessage** stream.

Parameters

Name	Type	Description
value	unsigned char*	Pointer to the value to be written.

Return Value

None.

writeBytes

Syntax

```
WriteBytesEx(value, offset, length)
```

Description

Writes a portion of a byte array (unsigned char values) to the **BytesMessage** stream. For example, to extract "nag" from "manager", set iOffset=2 and iLength=3.

Parameters

Name	Type	Description
pczValue	unsigned char*	Pointer to the value to be written.
offset	int	The initial offset within the byte array.
length	int	The number of bytes to use.

Return Value

None.

writeChar

Syntax

```
void writeChar(value)
```

Description

Writes an unsigned short integer to the **BytesMessage** stream.

Parameters

Name	Type	Description
value	unsigned short	The value to be written.

Return Value

None.

writeDouble

Syntax

```
void writeDouble(value)
```

Description

Writes a double numeric value to the **BytesMessage** stream.

Parameters

Name	Type	Description
value	double	The value to be written.

Return Value

None.

writeFloat

Syntax

```
writeFloat(value)
```

Description

Writes a floating-point numeric value to the **BytesMessage** stream.

Parameters

Name	Type	Description
value	float	The value to be written.

Return Value

None.

writeInt

Syntax

```
void writeInt(value)
```

Description

Writes an integer numeric value to the **BytesMessage** stream.

Parameters

Name	Type	Description
value	int	The value to be written.

Return Value

None.

writeLong

Syntax

```
void writeLong(value)
```

Description

Writes a long numeric value to the **BytesMessage** stream.

Parameters

Name	Type	Description
value	int	The value to be written.

Return Value

None.

writeShort

Syntax

```
void writeShort(value)
```

Description

Writes a short numeric value to the **BytesMessage** stream.

Parameters

Name	Type	Description
value	short	The value to be written.

Return Value

None.

10.2 The MapMessage Class

A MapMessage is used to send a set of name-value pairs, where names are Strings. The entries are accessed sequentially or randomly by name. The order of the entries is undefined. It inherits from Message, and adds a map message body.

The methods of the MapMessage Class are:

getBoolean

Syntax

```
STCBOOL getBoolean(name)
```

Description

The getBoolean method returns the boolean value with the given name

Parameters

Name	Type	Description
name	WString.	The name of the boolean property.

Return Value

STCBOOL

getByte

Syntax

```
unsigned char getByte(name)
```

Description

The getByte method returns the byte value with the given name.

Parameters

Name	Type	Description
name	WString	The name of the byte.

Return Value

unsigned char

getBytes

Syntax

```
unsigned char* getBytes(name)
```

Description

The getBytes method returns the byte array value with the given name as a variable.

Parameters

Name	Type	Description
name	WString	The name of the byte array value.

Return Value

unsigned char*
Pointer to the byte array value.

getBytes

Syntax

```
unsigned int getBytes(name, value, length)
```

Description

The getBytes method returns the int value with the given name as a variable.

Parameters

Name	Type	Description
name	WString	The name of the byte array value.
value	int	Value to be written.
length	int	The number of bytes to use.

Return Value

unsigned int
The value read.

getChar

Syntax

```
wchar_t getChar(name)
```

Description

The getChar property returns the Unicode character value with the given name.

Parameters

Name	Type	Description
name	WString	The name of the Unicode character.

Return Value

wchar_t

getDouble

Syntax

```
double getDouble(name)
```

Description

The getDouble method returns the double value with the given name.

Parameters

Name	Type	Description
name	WString	The name of the double property.

Return Value

double

getFloat

Syntax

```
float getFloat(name)
```

Description

The getFloat method returns the float value with the given name.

Parameters

Name	Type	Description
name	WString	The name of the float property.

Return Value

float

getInt

Syntax

```
int getInt(name)
```

Description

The getInt method returns the int value of the given name

Parameters

Name	Type	Description
name	WString	The value of the name.

Return Value

int

getLong

Syntax

```
int64_t getLong(name)
```

Description

The getLong method returns the long value of the given name.

Parameters

Name	Type	Description
name	WString	The long value of the name.

The GetObject Method

The GetProperty Method

The GetProperty method returns the Visual Basic data type property value with the given name, into the Message.

```
MapMessage.GetProperty(name As String)
```

Name	Description
name	Name of the currency property.

The GetShort Method

The GetShort method returns the short value with the given name.

```
MapMessage.GetShort (name As String) As Integer
```

Name	Description
name	The name of the short currency property.

The GetString Method

Return the String value with the given name

```
MapMessage.GetString(name As String) As String
```

Name	Description
name	The name of the String property.

The ItemExists Method

The ItemExists method checks to verify if an item exists in the MapMessage.

```
MapMessage.ItemExists(name As String) As Boolean
```

Name	Description
name	The name of the item to check.

The PropertyExists Method

The PropertyExists method checks if a property value exists.

```
MapMessage.PropertyExists (name As String) As Boolean
```

Name	Description
name	The name of the property value.

The SetBoolean Method

The SetBoolean method sets a boolean property value with the given name, into the Message.

```
MapMessage.SetBoolean (name As String, value As Boolean)
```

Name	Description
name	The name of the property value.
value	The value to set in the message.

The SetByte Method

The SetByte method sets a byte value with the given name, into the Map.

```
MapMessage.SetByte(name As String, value As Byte)
```

Name	Description
name	The name of the byte property.
value	The byte property value to set in the message.

The SetBytes Method

The SetBytes method sets a byte array or a portion of value with the given name, into the Map.

```
MapMessage.SetBytes(name As String, value, [offset], [length])
```

Name	Description
name	The name of the Bytes property.
value	The byte array value to set in the Map.
offset	The initial offset within the byte array.
length	The number of bytes to use.

The SetChar Method

The SetChar method sets a Unicode character value with the given name, into the Map.

```
MapMessage.SetChar(name As String, value As Integer)
```

Name	Description
name	The name of the Unicode character.
value	The Unicode character value to set in the Map.

The SetDouble Method

The SetDouble method sets a double value with the given name, into the Map.

```
MapMessage.SetDouble(name As String, value As Double)
```

Name	Description
name	The name of the double property.
value	The double property value to set in the map.

The SetFloat Methods

The SetFloat method sets a float value with the given name, into the Map.

```
MapMessage.SetFloat(name As String, value As Single)
```

Name	Description
name	The name of the float property.
value	The the float value to set in the map.

The SetInt Method

Set an long value with the given name, into the Map

```
MapMessage.SetInt(name As String, value As Long)
```

Name	Description
name	The name of the long property.
value	The long property value to set in the message.

The SetLong Method

The SetLong method sets a currency value with the given name, into the Map.

```
MapMessage.SetLong(name As String, value As Currency)
```

Name	Description
name	The name of the currency property.
value	The currency property value to set in the message.

The SetObject Method

This method is currently not supported.

The SetProperty Method

Sets a Visual Basic data type property value with the given name, into the Message.

```
MapMessage.SetProperty(name As String, value)
```

Name	Description
name	The name of the property.
value	The value to set.

The SetShort Method

The SetShort method sets a short value with the given name, into the Map.

```
MapMessage.SetShort(name As String, value As Integer)
```

Name	Description
name	The name of the short property.
value	The integer property value to set in the map.

The SetString Method

The SetString method sets a String value with the given name, into the Map.

```
MapMessage.SetString(name As String, value As String)
```

Name	Description
name	The name of the string property.
value	The string value to set into the map.

10.2.1. The TextMessage Class

A TextMessage is used to send a message containing text. It adds a text message body. When a client receives a TextMessage, it is in read-only mode. If an attempt is made to write to the message while in read-only mode, an error is returned. However, if ClearBody is called first, then message can be then read from and written to.

The TextMessage functions include the following:

GetText on page 339

SetText on page 340

SetText on page 340

GetText

Syntax

```
WString getText()
```

Description

Retrieves the string containing the data associated with the message.

Return Value

WString
Wide string.

SetText

Syntax

```
void SetText(buffer)
```

Description

Sets the string containing the data associated with the message.

Parameters

Name	Type	Description
Buffer	WString	The text string.

Return Value

None.

SetText

Syntax

```
void SetText(buffer)
```

Description

Sets the string containing the data associated with the message.

Parameters

Name	Type	Description
Buffer	char*	Pointer to the text string.

Return Value

None.

10.2.2. The Connection Interface for JMS in C++

A **Connection** object is an active connection to a JMS point-to-point provider or an active connection to a JMS pub/sub provider. A client uses a Connection to create one or more Sessions for producing and consuming messages.

The **Connection** interface includes the following methods:

close on page 341

getClientID on page 341

setClientID on page 341

start on page 342

stop on page 342

close

Syntax

```
void close()
```

Description

Closes the specified connection.

Return Value

None.

getClientID

Syntax

```
WString getClientID()
```

Description

Retrieves the client ID associated with the specified **Connection** object.

Return Value

WString*
WString (wide string) object containing the text.

setClientID

Syntax

```
void setClientID(ClientID)
```

Description

Sets the client ID to be associated with the specified **Connection** object. In other words, this method allows you to name the connection.

Parameters

Name	Type	Description
clientID	WString	Text string for the client ID.

Return Value

None.

---

start

Syntax

```
void start()
```

Description

Starts (or restarts) delivering incoming messages via the specified **Connection** object. If the connection is already started, the call is ignored without error.

Return Value

None.

---

stop

Syntax

```
void stop()
```

Description

Temporarily halts delivering incoming messages via the specified **Connection** object. If the connection is already stopped, the call is ignored without error.

Return Value

None.

10.2.3. The QueueConnection Interface for JMS in C++

A QueueConnection is an active connection to a JMS PTP provider. A client uses a QueueConnection to create one or more QueueSessions for producing and consuming messages.

The QueueConnection Interface methods:

createQueueSession on page 342

---

createQueueSession

Syntax

```
QueueSession* createQueueSession(transacted, acknowledgeMode)
```

Description

Creates a **Session** object.

Parameters

Name	Type	Description
transacted	SBYN_BOOL	Flag: If **true**, the session is transacted.
acknowledgeMode	int	Ignored if the session is transacted. If the session is not transacted, indicates whether the consumer or client acknowledges messages that it receives.

Return Value

SBYN_Session* pointer.

10.2.4. The Session Interface for JMS in C++

The Session Interface methods:

---

close

Syntax

```
void close()
```

Description

Closes the specified session.

Note: *Sessions should be closed when they are no longer needed.*

Return Value

None.

commit

Syntax

```
void commit()
```

Description

Commits all messages done in this transaction and releases any locks currently held.

Return Value

None.

getTransacted

Syntax

```
STCBOOL getTransacted()
```

Description

Queries whether the specified session is or is not transacted.

STCBOOL
Returns **true** if the session is transacted; otherwise, returns **false**.

recover

Syntax

```
void recover()
```

Description

Stops message delivery in the specified session, causes all messages that might have been delivered but not acknowledged to be marked as **redelivered**, and restarts message delivery with the oldest unacknowledged message. Note that redelivered messages need not be delivered in the exact order they were originally delivered.

Return Value

None.

rollback

Syntax

```
rollback(pSessn, iError, pczError)
```

Description

Rolls back any messages done in this transaction and releases any locks currently held.

Return Value

None.

bytesMessage

Syntax

```
BytesMessage* bytesMessage()
```

Description

Creates a **BytesMessage**— an object used to send a message containing a stream of uninterpreted bytes.

Return Value

BytesMessage*
Pointer to the created message object.

createTextMessage

Syntax

```
TextMessage createTextMessage()
```

Description

Creates an uninitialized **TextMessage**— an object used to send a message containing a string to be supplied.

Return Value

TextMessage
The created message object.

createTextMessage

Syntax

```
TextMessage createTextMessage(stringBuffer)
```

Description

Creates an initialized **TextMessage**— an object used to send a message containing the supplied string.

Parameters

Name	Type	Description
stringBuffer	WString	stringBuffer to the Text message.

Return Value

TextMessage

10.2.5. The TopicConnection Interface for JMS in C++

The TopicConnection Interface methods are:

createTopicSession

Description

```
ConnectionConsumer createTopicSession(transacted, acknowledgeMode)
```

Syntax

Create a TopicSession

Parameters

Name	Description
transacted	If true, session is transacted.
acknowledgeMode	**msAutoAcknowledge** = 1 : The session automatically acknowledges a client's receipt of a message when it has either successfully returned from a call to receive or the MessageListener it has called to process the message successfully returns. **msClientAcknowledge** = 2 : A client acknowledges a message by calling the message's acknowledge method. Acknowledging a consumed message automatically acknowledges the receipt of all messages that have been delivered by its session. **msDupsOkAcknowledge** = 3 : Instructs the session to lazily acknowledge the delivery of messages. This is likely to result in the delivery of some duplicate messages if the Message Service fails. It should only be used by consumers that are tolerant of duplicate messages. The benefit is the reduction of session overhead, achieved by minimizing the work done to prevent duplicates.

Return Value

ConnectionConsumer

close

Syntax

```
void close()
```

Description

Closes the specified session.

Note: *Sessions should be closed when they are no longer needed.*

Return Value

None.

---

getClientID

Syntax

```
WString getClientID()
```

Description

Retrieves the client ID associated with the specified **Connection** object.

Return Value

WString*
Pointer to a **WString** (wide string) object containing the text.

---

setClientID

Syntax

```
void setClientID(clientID)
```

Description

Sets the client ID to be associated with the specified **Connection** object. In other words, this method allows you to name the connection.

Parameters

Name	Type	Description
clientID	void	The text string for the client ID.

Return Value

None.

---

setClientID

Syntax

```
void setClientID(clientID)
```

Description

Sets the client ID to be associated with the specified **Connection** object. In other words, this method allows you to name the connection.

Parameters

Name	Type	Description
clientID	char*	Pointer to the text string for the client ID.

Return Value

char *
Pointer to the text string to the client ID.

---

getExceptionListener

Syntax

```
ExceptionListener getExceptionListener()
```

Description

Gets the `ExceptionListener` object for this connection.

Return Value

ExceptionListener

10.2.6. The QueueConnectionFactory Interface for JMS in C++

Using point-to-point messaging, a client uses a **QueueConnectionFactory** object to create **QueueConnection** objects.

The **QueueConnectionFactory** interface includes the following methods:

- **createQueueConnection** on page 348
- **createQueueConnection** on page 349

---

createQueueConnection

Syntax

```
QueueConnection* createQueueConnection()
```

Description

Constructs a **QueueConnection** object.

Return Value

QueueConnection*
Pointer to the **QueueConnection** object that was created.

createQueueConnection

Syntax

```
QueueConnection createQueueConnection(userName, password)
```

Description

Constructs a **Connection** object

Parameters

Name	Type	Description
userName	char*	Pointer to the userName.
password	char*	Pointer to the password.

Return Value

QueueConnection* pointer.

10.2.7. The TopicConnectionFactory Interface for JMS in C++

Using pub/sub messaging, a client uses a **TopicConnectionFactory** object to create **TopicConnection** objects.

The **TopicConnectionFactory** interface includes the following methods:

- **createTopicConnectionFactory** on page 349

createTopicConnectionFactory

Syntax

```
TopicConnection* createTopicConnectionFactory()
```

Description

Constructs a **TopicConnectionFactory** for the specified host and port. Once constructed, it can create **TopicConnection** objects for a pub/sub JMS provider.

Return Value

TopicConnection*
Pointer to the **TopicConnection** object that was created.

10.2.8. The ExceptionListener Interface for JMS in C++

If the JMS IQ manager detects a serious problem with a Connection object, it informs the Connection object's ExceptionListener, if one has been registered. It does this by calling the listener's onException method, passing it a JMSException argument describing the problem.

This allows a client to be asynchronously notified of a problem. Some Connections only consume messages so they would have no other way to learn their Connection has failed.

A JMS provider should attempt to resolve connection problems themselves prior to notifying the client of them.

The ExceptionListener Interface methods:

OnException on page 350

OnException

Syntax

```
void OnException(exception)
```

Description

Notifies user of a JMS exception.

Parameters

Name	Description
exception	The JMS exception.

Returns

None.

10.2.9. The DeliveryMode Interface for JMS in C++

The delivery modes supported by the JMS API are PERSISTENT and NON_PERSISTENT.

A client marks a message as persistent if it feels that the application will have problems if the message is lost in transit. A client marks a message as non-persistent if an occasional lost message is tolerable. Clients use delivery mode to tell the JMS IQ manager how to balance message transport reliability throughput.

Delivery mode only covers the transport of the message to its destination. Retention of a message at the destination until its receipt is acknowledged is not guaranteed by a PERSISTENT delivery mode. Clients should assume that message retention policies are set administratively. Message retention policy governs the reliability of message delivery from destination to message consumer. For example, if a client's message storage space is exhausted, some messages as defined by a site specific message retention policy may be dropped.

A message is guaranteed to be delivered once-and-only-once by a JMS Provider if the delivery mode of the message is persistent and if the destination has a sufficient message retention policy.

NON_PERSISTENT Field

This is the lowest overhead delivery mode because it does not require that the message be logged to stable storage. The level of JMS provider failure that causes a NON_PERSISTENT message to be lost is not defined.

A JMS provider must deliver a NON_PERSISTENT message with an at-most-once guarantee. This means it may lose the message but it must not deliver it twice.

```
public static final int NON_PERSISTENT
```

PERSISTENT Field

This mode instructs the JMS provider to log the message to stable storage as part of the client's send operation. Only a hard media failure should cause a PERSISTENT message to be lost.

10.2.10. The Queue Interface for JMS in C++

A Queue object encapsulates a provider-specific queue name. In this manner, a client specifies the identity of queue to JMS methods. The actual length of time messages are held by a queue and the consequences of resource overflow are not defined by JMS.

The Queue Interface methods are:

getQueueName on page 351

toString on page 351

---

getQueueName

Syntax

```
WString getQueueName()
```

Description

Get the name of this queue. Clients that depend upon the name, are not portable.

Returns

WString
Wide string object.

---

toString

Syntax

```
WString toString()
```

Description

Return a pretty printed version of the queue name

Returns

WString
Wide string object.

10.2.11. The TemporaryQueue Interface for JMS in C++

A TemporaryQueue is a unique Queue object created for the duration of a QueueConnection. It is a system defined queue that can only be consumed by the QueueConnection that created it.

The TemporaryQueue Interface methods are:

Delete on page 352

Delete

Syntax

```
void Delete()
```

Description

Delete this temporary queue. If there are still existing senders or receivers still using it, then a JMSException will be thrown.

Throws JMSException if JMS implementation fails to delete a Temporary topic due to some internal error.

Returns

Nothing.

10.2.12. The Topic Interface for JMS in C++

A Topic object encapsulates a provider-specific topic name. The topic object provides the means for a client to specify the identity of a topic to JMS methods.

Many Pub/Sub implementations group topics into hierarchies and provide various options for subscribing to parts of the hierarchy. JMS places no restriction on what a Topic object represents.

The Topic Interface methods:

getTopicName on page 352

toString on page 353

getTopicName

Syntax

```
WString getTopicName()
```

Description

Gets the name of this topic.

Returns

WString
Wide string object.

toString

Syntax

WString **toString**()

Description

Returns a string representation of this object.

Returns

WString
Wide string object.

10.2.13. The TemporaryTopic Interface for JMS in C++

A TemporaryTopic object is a unique Topic object created for the duration of a TopicConnection. It is a system-defined topic that can be consumed only by the TopicConnection that created it.

The TemporaryTopic Interface methods are:

Delete on page 353

Delete

Syntax

```
void Delete()
```

Description

Deletes this temporary topic. If there are existing subscribers still using it, a `JMSException` will be thrown.

Returns

None.

10.2.14. The MessageProducer Interface for JMS in C++

The **MessageProducer** interface includes the following methods:

- **close ON PAGE 354**
- **getDeliveryMode ON PAGE 354**
- **getDisableMessageID** on page 354
- **getDisableMessageTimestamp** on page 355
- **getJMS_ProducerID** on page 355
- **getPriority** on page 355
- **getTimeToLive** on page 355

- **setDeliveryMode** on page 356
- **setDisableMessageID** on page 356
- **getDisableMessageTimestamp** on page 355
- **setJMS_ProducerID** on page 357
- **setPriority** on page 357
- **setTimeToLive** on page 358

close

Syntax

```
void close()
```

Description

Closes the specified message producer.

Note: *When a message producer is no longer needed, it should be closed.*

Return Value

None.

getDeliveryMode

Syntax

```
int QueueSenderGetDeliveryMode()
```

Description

Retrieves the value of the **DeliveryMode** property of the specified message producer.

Return Value

int

getDisableMessageID

Syntax

```
STCBOOL getDisableMessageID()
```

Description

Queries whether message IDs are or are not disabled for the specified message producer.

Return Value

SBYN_BOOL
Returns **true** if message IDs are disabled; otherwise, returns **false**.

getDisableMessageTimestamp

Syntax

```
STCBOOL getDisableMessageTimestamp()
```

Description

Queries whether message timestamping is or is not disabled for the specified message producer.

Return Value

SBYN_BOOL
Returns **true** if message timestamping is disabled; otherwise, returns **false**.

getJMS_ProducerID

Syntax

```
void getJMS_ProducerID(ProducerID)
```

Description

Retrieves the value of the **ProducerID** property for the specified message producer.

Return Value

None.

getPriority

Syntax

```
int getPriority()
```

Description

Queries the value of the message **Priority** property of the specified message producer.

Return Value

int
Message priority level, from **0** (least expedited) through **9** (most expedited).

getTimeToLive

Syntax

```
int64_t getTimeToLive()
```

Description

Queries the value of the **TimeToLive** property of the specified message producer.

Return Value

int64_t
Length of time (milliseconds elapsed since dispatch time) that a produced message should be retained by JMS.

setDeliveryMode

Syntax

```
void setDeliveryMode(DeliveryMode)
```

Description

Sets the value of the **DeliveryMode** property of the specified message producer.

Parameters

Name	Type	Description
DeliveryMode	int	Value for the **DeliveryMode** property.

Return Value

None.

setDisableMessageID

Syntax

```
void setDisableMessageID(value)
```

Description

Determines whether message IDs are disabled for this queue sender. Default **false**.

Parameters

Name	Type	Description
value	STC_BOOL	Flag, default **false**; if **true**, message IDs are disabled.

Return Value

None.

setDisableMessageTimestamp

Syntax

```
void setDisableMessageTimestamp(value)
```

Description

Determines whether message timestamping is disabled for this message producer. Default **false**.

Since message timestamps take effort to create and increase the size of a message, this flag can be set **true** to reduce overhead if message IDs are not used by an application.

Parameters

Name	Type	Description
value	STC_BOOL	Flag, default **false**; if **true**, message timestamping is disabled.

Return Value

None.

setJMS_ProducerID

Syntax

```
void setJMS_ProducerID(ProducerID)
```

Description

Sets the value of the **ProducerID** property for the specified message producer.

Parameters

Name	Type	Description
ProducerID	char*	Pointer to text string containing the Producer ID.

Return Value

None.

setPriority

Syntax

```
void setPriority(deliveryMode)
```

Description

Sets the value of the message **Priority** property, from **0** (least expedited) through **9** (most expedited).

Parameters

Name	Type	Description
deliverMode	int	Message priority level.

Return Value

None.

---

setTimeToLive

Syntax

```
void setTimeToLive(TimeToLive)
```

Description

Sets the value of the **TimeToLive** property of the specified message producer.

Parameters

Name	Type	Description
TimeToLive	long	Value to be used for TimeToLive: Length of time (milliseconds elapsed since dispatch time) that a produced message should be retained by JMS.

Return Value

None.

10.2.15. The QueueSender Interface for JMS in C++

Using point-to-point messaging, a client uses a **QueueSender** object to send messages to a queue. After sending a message to a queue, a client may retain the message and modify it without affecting the message that has been sent. The same message object may be sent multiple times.

The **QueueSender** interface includes the following methods:

- **send** ON PAGE 358
- **send** on page 359
- **send** on page 359
- **send** on page 360
- **send** on page 360
- **send** on page 361
- **send** on page 361
- **send** on page 362

---

send

Syntax

```
void send(message)
```

Description

Sends the specified message to the queue.

Parameters

Name	Type	Description
message	Message*	Pointer to the message to send.

Return Value

None.

send

Syntax

```
void send(message)
```

Description

Sends the specified message to the queue.

Parameters

Name	Type	Description
message	BytesMessage*	Pointer to the message to send.

Return Value

None.

send

Syntax

```
void send(message)
```

Description

Sends the specified message to the queue.

Parameters

Name	Type	Description
message	TextMessage*	Pointer to the message to send.

Return Value

None.

send

Syntax

```
void send(message, DeliveryMode, priority, timeToLive)
```

Description

Sends the specified message to the queue, overriding one or more default values for properties of the specified queue sender.

Parameters

Name	Type	Description
message	message*	Pointer to the message to send.
deliveryMode	int	Value to be used for **DeliveryMode.**
priority	int	Value to be used for **Priority**.
timeToLive	int64_t	Value to be used for **TimeToLive**: Length of time (milliseconds elapsed since dispatch time) that a produced message should be retained by JMS.

Return Value

None.

send

Syntax

```
void send(message, DeliveryMode, priority, timeToLive)
```

Description

Sends the specified message to the queue, overriding one or more default values for properties of the specified queue sender.

Parameters

Name	Type	Description
message	BytesMessage*	Pointer to the message to send.
deliveryMode	int	Value to be used for **DeliveryMode.**
priority	int	Value to be used for **Priority**.
timeToLive	int64_t	Value to be used for **TimeToLive**: Length of time (milliseconds elapsed since dispatch time) that a produced message should be retained by JMS.

Return Value

None.

send

Syntax

```
void send(message, DeliveryMode, priority, timeToLive)
```

Description

Sends the specified message to the queue, overriding one or more default values for properties of the specified queue sender.

Parameters

Name	Type	Description
message	TextMessage*	Pointer to the message to send.
deliveryMode	int	Value to be used for **DeliveryMode.**
priority	int	Value to be used for **Priority**.
timeToLive	int64_t	Value to be used for **TimeToLive**: Length of time (milliseconds elapsed since dispatch time) that a produced message should be retained by JMS.

Return Value

None.

send

Syntax

```
void send(queue, message)
```

Description

Sends the specified message to the specified queue, using default values for properties (**DeliveryMode**, **Priority**, and **TimeToLive**) of the specified queue sender.

Typically, a message producer is assigned a queue at creation time; however, the JMS API also supports unidentified message producers, which require that the queue be supplied every time a message is sent.

Parameters

Name	Type	Description
queue	Queue*	Pointer to the **QueueSender** object.
message	Message*	Pointer to the message to send.

Return Value

None.

send

Syntax

```
send(queue, message, deliveryMode, priority, timeToLive)
```

Description

Sends the specified message to the specified queue, overriding one or more default values for properties of the specified queue sender.

Parameters

Name	Type	Description
queue	Queue*	Pointer to the **QueueSender** object.
message	Message*	Pointer to the message to send.
deliveryMode	int	Value to be used for **DeliveryMode.**
priority	int	Value to be used for **Priority**.
timeToLive	int64_t	Value to be used for **TimeToLive**: Length of time (milliseconds elapsed since dispatch time) that a produced message should be retained by JMS.

Return Value

None.

10.2.16. The TopicPublisher Interface

The **TopicPublisher** interface includes the following methods:

getTopic

Syntax

```
Topic* getTopic()
```

Description

Gets the specified topic.

Returns

Topic*
Pointer to the specified topic.

---

publish

Syntax

```
void publish(message)
```

Description

Publishes the specified message to the topic, using default values for properties (**DeliveryMode**, **Priority**, and **TimeToLive**) of the specified topic publisher.

Parameters

Name	Type	Description
message	Message*	Pointer to the message to be published.

Return Value

None.

---

publish

Syntax

```
void publish(message)
```

Description

Publishes the specified message to the topic, using default values for properties (**DeliveryMode**, **Priority**, and **TimeToLive**) of the specified topic publisher.

Parameters

Name	Type	Description
message	BytesMessage*	Pointer to the message to be published.

Return Value

None.

publish

Syntax

```
void publish(message)
```

Description

Publishes the specified message to the topic, using default values for properties (**DeliveryMode**, **Priority**, and **TimeToLive**) of the specified topic publisher.

Parameters

Name	Type	Description
message	TextMessage*	Pointer to the message to be published.

Return Value

None.

publish

Syntax

```
void publish(message, deliveryMode, priority, timeToLive)
```

Description

Publishes the specified message to the topic, overriding one or more default values for properties of the specified topic publisher.

Parameters

Name	Type	Description
message	Message*	Pointer to the message to be published.
deliveryMode	int	Value to be used for **DeliveryMode**.
priority	int	Value to be used for **Priority**.
timeToLive	int64_t	Value to be used for **TimeToLive**: Length of time (milliseconds elapsed since dispatch time) that a produced message should be retained by JMS.

Return Value

None.

publish

Syntax

```
void publish(message, deliveryMode, priority, timeToLive)
```

Description

Publishes the specified message to the topic, overriding one or more default values for properties of the specified topic publisher.

Parameters

Name	Type	Description
message	BytesMessage*	Pointer to the message to be published.
deliveryMode	int	Value to be used for **DeliveryMode**.
priority	int	Value to be used for **Priority**.
timeToLive	int64_t	Value to be used for **TimeToLive**: Length of time (milliseconds elapsed since dispatch time) that a produced message should be retained by JMS.

Return Value

None.

publish

Syntax

```
void publish(message, deliveryMode, priority, timeToLive)
```

Description

Publishes the specified message to the topic, overriding one or more default values for properties of the specified topic publisher.

Parameters

Name	Type	Description
message	TextMessage*	Pointer to the message to be published.
deliveryMode	int	Value to be used for **DeliveryMode**.
priority	int	Value to be used for **Priority**.
timeToLive	int64_t	Value to be used for **TimeToLive**: Length of time (milliseconds elapsed since dispatch time) that a produced message should be retained by JMS.

Return Value

None.

publish

Syntax

```
void publish(topic, message)
```

Description

Publishes the specified message to the specified topic.

Parameters

Name	Type	Description
topic	Topic*	Pointer to the **TopicPublisher** object.
message	Message*	Pointer to the message.

Return Value

None.

---

publish

Syntax

```
void publish(topic, message, deliveryMode, priority, timeToLive)
```

Description

Publishes the specified message to the specified topic, overriding one or more default values for properties of the specified topic publisher.

Parameters

Name	Type	Description
topic	Topic*	Pointer to the **TopicPublisher** object.
message	Message*	Pointer to the message to publish.
delivery	int	Value to be used for **DeliveryMode**.
priority	int	Value to be used for **Priority**.
timeToLive	long	Value to be used for **TimeToLive**: Length of time (milliseconds elapsed since dispatch time) that a produced message should be retained by JMS.

Return Value

None.

10.2.17. The QueueSession Interface for JMS in C++

The QueueSession Interface methods are:

createQueue on page 367

createReceiver on page 367

createReceiver on page 367

createSender on page 368

createTemporaryQueue on page 368

createQueue

Syntax

```
queue createQueue(queueName)
```

Description

Creates an identity with a specific queue name; does not create a physical queue.

This functionality is provided for rare cases where clients need to dynamically create a queue identity with a provider-specific name. Clients that depend on this functionality are not portable.

Parameters

Name	Type	Description
queueName	WString	The name of the queue.

Return Value

WString.

createReceiver

Syntax

```
QueueReceiver* createReceiver(queue)
```

Description

Creates a **Receiver** object to receive messages;

Parameters

Name	Type	Description
queue	queue*	Pointer to the queue.

Return Value

QueueReceiver* pointer.

createReceiver

Syntax

```
QueueReceiver* createReceive(queue, messageSelector)
```

Description

Creates a **Receiver** object to receive messages using a message selector.

Parameters

Name	Type	Description
queue	Queue*	Pointer to the queue.
messageSelector	char*	Pointer to the text of the message selector.

Return Value

QueueReceiver* pointer.

---

createSender

Syntax

```
QueueSender createSender(queue)
```

Description

Creates a **Sender** object to send messages.

Parameters

Name	Type	Description
queue	queue*	Pointer to the queue.

Return Value

QueueSender* pointer.

---

createTemporaryQueue

Syntax

```
TemporaryQueue createTemporaryQueue()
```

Description

Creates a **Temporary** object for a specified session.

Return Value

TemporaryQueue.

10.2.18. The TopicSession Interface for JMS in C++

The TopicSession Interface methods are:

createDurableSubscriber on page 369

createDurableSubscriber on page 370

createPublisher on page 370

createSubscriber on page 371

createDurableSubscriber

Syntax

```
TopicSubscriber* createDurableSubscriber(topic, name)
```

Description

Creates a durable subscriber to the specified topic, specifying whether messages published by its own connection should be delivered to it.

Using pub/sub messaging, if a client needs to receive all the messages published on a topic, including messages published while the subscriber is inactive, it uses a *durable subscriber*. The JMS provider retains a record of this durable subscription and ensures that each message from the topic's publishers is retained until either it has been acknowledged by this durable subscriber or else it has expired.

Sessions with durable subscribers must always provide the same client ID, and each client must specify a name that (within the given client ID) uniquely identifies each durable subscription it creates. Only one session at a time can have a TopicSubscriber for a particular durable subscription. An *inactive* durable subscriber is one that exists but does not currently have a message consumer associated with it.

A client can change an existing durable subscription by creating a durable TopicSubscriber with the same name and a new topic (and/or *message selector*). Changing a durable subscriber is equivalent to deleting the old one and creating a new one.

Parameters

Name	Type	Description
topic	Queue*	Pointer to the topic.
name	char*	Pointer to the text string containing the client ID of the durable subscriber.

Return Value

TopicSubscriber*

Pointer to the created **TopicSubscriber** object.

createDurableSubscriber

Syntax

```
TopicSubscriber* createDurableSubscriber(topic, name,
messageSelector, noLocal)
```

Description

Creates a durable subscriber to the specified topic, using a message selector (*messageSelector*) and/or specifying whether messages published by its own connection should be delivered to it (*noLocal*).

Parameters

Name	Type	Description
topic	Topic*	Pointer to the **topic**.
name	char*	Pointer to the text string containing the client ID of the durable subscriber.
messageSelector	char*	Pointer to the string containing the text of the message selector; only messages with properties matching the message selector expression are delivered. A value of null or an empty string indicates that there is no message selector for the message consumer.
noLocal	STCBOOL	Flag, default **false**. If **true**, inhibits the delivery of messages published by its own connection.

Return Value

TopicSubscriber*
Pointer to the created **TopicSubscriber** object.

createPublisher

Syntax

```
TopicPublisher createPublisher(topic)
```

Description

Creates a publisher for the specified topic.

Parameters

Name	Type	Description
topic	Topic*	Pointer to the **topic**.

Return Value

TopicPublisher*
Pointer to the created **TopicPublisher** object.

createSubscriber

Syntax

```
TopicSubscriber* createSubscriber(topic)
```

Description

Creates a *nondurable* subscriber to the specified topic—in other words, a subscriber that receives only those messages that are published while the subscriber is active.

Parameters

Name	Type	Description
topic	Topic*	Pointer to the **topic**.

Return Value

TopicSubscriber*
Pointer to the **TopicSubscriber** object.

createSubscriber

Syntax

```
TopicSubscriber* createSubscriber(topic, messageSelector, noLocal)
```

Description

Creates a *nondurable* subscriber to the specified topic—in other words, a subscriber that receives only those messages that are published while the subscriber is active

In some cases, a connection may both publish and subscribe to a topic. The NoLocal parameter allows a subscriber to inhibit the delivery of messages published by its own connection. The default value for this attribute is **false**.

Parameters

Name	Type	Description
topic	Topic*	Pointer to the **topic**.
messageSelector	char*	Pointer to the string containing the text of the message selector; only messages with properties matching the message selector expression are delivered. A value of null or an empty string indicates that there is no message selector for the message consumer.
noLocal	STCBOOL	Flag, default **false**. If **true**, inhibits the delivery of messages published by its own connection.

Return Value

TopicSubscriber*
Pointer to the **TopicSubscriber** object.

createTemporaryTopic

Syntax

```
TemporaryTopic* createTemporaryTopic()
```

Description

Creates a temporary topic that lives only as long as the specified TopicSession does (unless the topic is deleted earlier).

Return Value

TemporaryTopic*
Pointer to the created **TemporaryTopic** object.

createTopic

Syntax

```
Topic createTopic(topicName)
```

Description

Creates a topic identity with a specific topic name; does ***not*** create a physical topic.

This functionality is provided for rare cases where clients need to dynamically create a topic identity with a provider-specific name. Clients that depend on this functionality are not portable.

Parameters

Name	Type	Description
topicName	WString	The text string containing the name of the topic.

Return Value

Topic

unsubscribe

Syntax

```
void unsubscribe(name)
```

Description

Unsubscribes a durable subscription that has been created by a client. Note that it is an error to delete a durable subscription while there is an active TopicSession for the subscription, or while a consumed message is part of a pending transaction or has not been acknowledged in the session.

Parameters

Name	Type	Description
name	void	The name used to identify this subscription.

Return Value

None.

10.2.19. The Xid Interface for JMS in C++

The Xid Interface has the following methods:

getBranchQualifier on page 373

getFormatId on page 373

getGlobalTransactionId on page 374

---

getBranchQualifier

Syntax

```
const unsigned char* getBranchQualifier(&pl)
```

Description

Obtain the transaction branch identifier part of XID as an array of bytes.

Parameters

Name	Type	Description
&pl	long	Returns the long address.

Return Value

const unsigned char*

---

getFormatId

Syntax

```
uint64_t getFormatId()
```

Description

Obtain the format identifier part of the XID.

Return Value

uint64_t

---

getGlobalTransactionId

Syntax

```
const unsigned char* getGlobalTransactionId(&pl)
```

Description

Obtain the global transaction identifier part of XID as an array of bytes.

Parameters

Name	Type	Description
&pl	long	Returns the long address.

Return Value

const unsigned char*

10.2.20. The XAResource Interface for JMS in C++

The XAResource Interface has the following methods:

Flag definitions

TMENDRSCAN	End a recovery scan
TMFAIL	Disassociates the caller and mark the transaction branch rollback-only.
TMJOIN	Caller is joining existing transaction branch.
TMNOFLAGS	Use TMNOFLAGS to indicate no flags value is selected.
TMONEPHASE	Caller is using one-phase optimization.
TMRESUME	Caller is resuming association with suspended transaction branch.
TMSTARTRSCAN	Start a recovery scan.
TMSUCCESS	Disassociate caller from transaction branch
TMSUSPEND	Caller is suspending (not ending) association with transaction branch.
XA_OK	The transaction work has been prepared normally.
XA_RDONLY	The transaction branch has been read-only and has been committed.

commit

Syntax

```
void commit(Xid, onePhase)
```

Description

Commits the global transaction specified by xid

Parameters

Name	Type	Description
xid	Xid*	A pointer to the global transaction identifier.
onePhase	STCBOOL	Flag; if true, a one-phase commit protocol is used to commit the work done on behalf of xid.

Returns

void.

Xid**recover

Syntax

```
void Xid**recover(flag)
```

Description

This method is used during recovery to obtain the list of transaction branches that are currently in prepared or heuristically completed states.

Parameters

Name	Type	Description
flag	int	One of TMSTARTRSCAN, TMENDRSCAN, TMNOFLAGS. TMNOFLAGS must be used when no other flags are set in flags.

Returns

int.

rollback

Syntax

```
void rollback(xid)
```

Description

Roll back work done on behalf of a transaction branch.

Parameters

Name	Type	Description
xid	Xid*	A global transaction identifier

Returns

None.

getTransactionTimeout

Syntax

```
int getTransactionTimeout()
```

Description

Obtain the current transaction timeout value set for this XAResource instance. If `XAResource.setTransactionTimeout` was not used prior to calling this method, the return value is the default timeout set for the resource manager; otherwise, the value used in the previous `setTransactionTimeout` call is returned.

Returns

None.

setTransactionTimeout

Syntax

```
STCBOOL setTransactionTimeout()
```

Description

Sets the current transaction timeout value for this `XAResource` instance. Once set, this timeout value is effective until `setTransactionTimeout` is invoked again with a different value. To reset the timeout value to the default value used by the resource manager, set the value to zero. If the timeout operation is performed successfully, the method returns *true*; otherwise *false*. If a resource manager does not support transaction timeout value to be set explicitly, this method returns *false*

Returns

STCBOOL

---

isSameRM

Syntax

int isSameRM(xares)

Description

This method is called to determine if the resource manager instance represented by the target object is the same as the resource manager instance represented by the parameter *xares*.

Parameters

Name	Type	Description
xid	Xid*	A global transaction identifier

Returns

int

---

prepare

Syntax

```
Xid* prepare(xid)
```

Description

Asks the resource manager to prepare for a transaction commit of the transaction specified in xid.

Parameters

Name	Type	Description
xid	Xid*	A global transaction identifier

Returns

int

A value indicating the resource manager's decision on the outcome of the transaction. The possible values are XA_RDONLY or XA_OK.

start

Syntax

```
void start(xid, flags)
```

Description

Start work on behalf of a transaction branch specified in xid. If TMJOIN is specified, the start is for joining a transaction previously seen by the resource manager. If TMRESUME is specified, the start is to resume a suspended transaction specified in the parameter xid. If neither TMJOIN or TMRESUME is specified and the transaction specified by xid has previously been seen by the JMS, the resource manager throws the XAException exception.

Parameters

Name	Type	Description
xid	Xid*	A global transaction identifier
flags	int	One of TMNOFLAGS, TMJOIN, or TMRESUME

Returns

None

end

Syntax

```
void end(xid, flags)
```

Description

Ends the work performed on behalf of a transaction branch. The JMS IQ manager disassociates the XA resource from the transaction branch specified and allows the transaction be completed.

If TMSUSPEND is specified in flags, the transaction branch is temporarily suspended in an incomplete state. The transaction context must then be resumed by specifying TMRESUME.

If TMFAIL is specified, the message failed. The JMS IQ manager may mark the transaction as rollback-only.

If TMSUCCESS is specified, the message has completed successfully.

Parameters

Name	Type	Description
xid	Xid*	A global transaction identifier
flags	int	One of TMSUCCESS, TMFAIL, or TMSUSPEND

Returns

None.

10.2.21. MSGSRVC_API *Lookup

The methods for the MSGSRVC_API *Lookup are:

*LookupQueueConnectionFactory

Syntax

```
*LookupQueueConnectionFactory(dllname, initString, hostname, port,
usPortOffset, iMaxRetires)
```

Description

Constructs a QueueConnectionFactory for the specified host and port using the given <iniString> name. You can replace this method if you are using your own naming service lookup API call.

Parameters

Name	Type	Description
dllname	const char*	dllname should always be set to MSCLIENT_DLL_NAME unless you rename the default libraries shipped with the API Kit.
initString	const char*	User defined name.
hostName	const char*	The default is DEFAULT_SERVER_NAME if you do not define a name.
port	unsigned short	The default is DEFAULT_PORT if you do not define a port.
usPortOffset	unsigned short	The default port offset is 0 if you do not define an offset.
iMaxRetries	int	The default number of maximum retries is 30 if you do not define a maximum number of retries.

Returns

QueueConnectionFactory

*LookupXAQueueConnectionFactory

Syntax

```
*LookupXAQueueConnectionFactory(dllname, initString, hostname, port,
usPortOffset, iMaxRetires)
```

Description

Constructs a QueueXAConnectionFactory for the specified host and port using the given <iniString> name. You can replace this method if you are using your own naming service lookup API call.

Parameters

Name	Type	Description
dllname	const char*	dllname should always be set to MSCLIENT_DLL_NAME unless you rename the default libraries shipped with the API Kit.
initString	const char*	User defined name.
hostName	const char*	The default is DEFAULT_SERVER_NAME if you do not define a name.
port	unsigned short	The default is DEFAULT_PORT if you do not define a port.
usPortOffset	unsigned short	The default port offset is 0 if you do not define an offset.
iMaxRetries	int	The default number of maximum retries is 30 if you do not define a maximum number of retries.

Returns

XAQueueConnectionFactory

*LookupQueue

Syntax

```
*LookupQueue(const char*, const char*)
```

Description

Constructs a topic using the given topicName.

Parameters

Name	Type	Description
User defined	const char*	User defined string.
User defined	const char*	User defined string.

Returns

Queue

*LookupTopicConnectionFactory

Syntax

```
*LookupTopicConnectionFactory(dllname, initString, hostname, port,
usPortOffset, iMaxRetires)
```

Description

Constructs a TopicConnectionFactory for the specified host and port using the given <iniString> name. You can replace this method if you are using your own naming service lookup API call.

Parameters

Name	Type	Description
dllname	const char*	dllname should always be set to MSCLIENT_DLL_NAME unless you rename the default libraries shipped with the API Kit.
initString	const char*	User defined name.
hostName	const char*	The default is DEFAULT_SERVER_NAME if you do not define a name.
port	unsigned short	The default is DEFAULT_PORT if you do not define a port.
usPortOffset	unsigned short	The default port offset is 0 if you do not define an offset.
iMaxRetries	int	The default number of maximum retries is 30 if you do not define a maximum number of retries.

Returns

TopicConnectionFactory

---

*LookupXATopicConnectionFactory

Syntax

```
*LookupXATopicConnectionFactory(dllname, initString, hostname, port,
usPortOffset, iMaxRetires)
```

Description

Constructs a XATopicConnectionFactory for the specified host and port using the given <iniString> name. You can replace this method if you are using your own naming service lookup API call.

Parameters

Name	Type	Description
dllname	const char*	dllname should always be set to MSCLIENT_DLL_NAME unless you rename the default libraries shipped with the API Kit.
initString	const char*	User defined name.
hostName	const char*	The default is DEFAULT_SERVER_NAME if you do not define a name.
port	unsigned short	The default is DEFAULT_PORT if you do not define a port.
usPortOffset	unsigned short	The default port offset is 0 if you do not define an offset.
iMaxRetries	int	The default number of maximum retries is 30 if you do not define a maximum number of retries.

Returns

XATopicConnectionFactory

*LookupTopic

Syntax

```
*LookupTopic(dllname, topicName)
```

Description

Constructs a topic using the given topicName.

Parameters

Name	Type	Description
dllname	const char*	dllname should always be set to MSCLIENT_DLL_NAME unless you rename the default libraries shipped with the API Kit.
topicName	const char*	Name of the topic.

Returns

Topic

*CreateXid

Syntax

```
*CreateXid(dllname)
```

Description

Constructs an Xid.

Parameters

Name	Type	Description
dllname	const char*	dllname should always be set to MSCLIENT_DLL_NAME unless you rename the default libraries shipped with the API Kit.

Returns

Xid

*LookupXADataSource

Syntax

```
*LookupXADataSource(const char*, const char*)
```

Description

Used to retrieve the XADataSource using a thirdparty library. For example, call LookupXADataSource(ORACLE_DLL_Name, “myname”) to get and instance of XADataSource

Parameters

Name	Type	Description
User defined	const char*	User defined String.
User defined	const char*	User defined String.

Returns

XADataSource

*LookupQueueConnectionFactoryExt

Syntax

```
*LookupQueueConnectionFactoryExt(dllname, initString, hostname, port,
usPortOffset, iMaxRetires, iInterval)
```

Description

Constructs QueueConnectionFactoryExt for the specified host and port using the given <iniString> name. You can replace this method if you are using your own naming service lookup API call.

Parameters

Name	Type	Description
dllname	const char*	dllname should always be set to MSCLIENT_DLL_NAME unless you rename the default libraries shipped with the API Kit.
initString	const char*	User defined name.
hostName	const char*	The default is DEFAULT_SERVER_NAME if you do not define a name.
port	unsigned short	The default is DEFAULT_PORT if you do not define a port.
usPortOffset	unsigned short	The default port offset is 0 if you do not define an offset.
iMaxRetries	int	The default number of maximum retries is 30 if you do not define a maximum number of retries.
iInterval	int	The default retry interval is 100 millisecond if you do not define the retry interval.

Returns

QueueConnectionFactory

*LookupXAQueueConnectionFactoryExt

Syntax

```
*LookupXAQueueConnectionFactoryExt(dllname, initString, hostname,
port, usPortOffset, iMaxRetires, iInterval)
```

Description

Constructs XAQueueConnectionFactoryExt for the specified host and port using the given <iniString> name. You can replace this method if you are using your own naming service lookup API call.

Parameters

Name	Type	Description
dllname	const char*	dllname should always be set to MSCLIENT_DLL_NAME unless you rename the default libraries shipped with the API Kit.
initString	const char*	User defined name.
hostName	const char*	The default is DEFAULT_SERVER_NAME if you do not define a name.
port	unsigned short	The default is DEFAULT_PORT if you do not define a port.
usPortOffset	unsigned short	The default port offset is 0 if you do not define an offset.
iMaxRetries	int	The default number of maximum retries is 30 if you do not define a maximum number of retries.
iInterval	int	The default retry interval is 100 millisecond if you do not define the retry interval.

Returns

XAQueueConnectionFactory

*LookupXATopicQueueConnectionFactoryExt

Syntax

```
*LookupXATopicConnectionFactoryExt(dllname, initString, hostname,
port, usPortOffset, iMaxRetires, iInterval)
```

Description

Constructs XATopicConnectionFactoryExt for the specified host and port using the given <iniString> name. You can replace this method if you are using your own naming service lookup API call.

Parameters

Name	Type	Description
dllname	const char*	dllname should always be set to MSCLIENT_DLL_NAME unless you rename the default libraries shipped with the API Kit.
initString	const char*	User defined name.
hostName	const char*	The default is DEFAULT_SERVER_NAME if you do not define a name.
port	unsigned short	The default is DEFAULT_PORT if you do not define a port.
usPortOffset	unsigned short	The default port offset is 0 if you do not define an offset.
iMaxRetries	int	The default number of maximum retries is 30 if you do not define a maximum number of retries.
iInterval	int	The default retry interval is 100 millisecond if you do not define the retry interval.

Returns

XATopicQueueConnectionFactory

Chapter 11

HP NonStop APIs

This chapter provides a detailed discussion of the Application Program Interfaces (APIs) for HP NonStop Server.

See **"Installing the eGate API Kit on HP NonStop Server" on page 15** for instructions on installing the eGate API Kit on HP NonStop Server.

For Java

- **The Standard Specification for the HP NonStop JMS API** on page 388

For C/C++

- **The Standard Specification for the HP NonStop C/C++ API** on page 388

11.1 The HP NonStop JMS and C/C++ APIs for the SeeBeyond Message Service

Refer to the following information on implementing JMS and C/C++ APIs:

- **Setting up the Java Client** on page 18
- **Setting up the C and C++ Client** on page 27

11.1.1. The Standard Specification for the HP NonStop JMS API

The HP NonStop JMS 2.0 API specification defined by Hewlett Packard is available for download:

"NonStop Server for Java Message Server User's Manual"

11.1.2. The Standard Specification for the HP NonStop C/C++ API

The HP NonStop C/C++ API specification defined by Hewlett Packard is available for download:

"**Introduction to HP C and C++ for NonStop Server"**

11.1.3. Setting up the HP NonStop Java Message Service

Using the **NonStop Server for Java Message Service User's Manual (522356-002)**, Section 2, *Install and Configure NSJMS* install and configure your NonStop JMS server. Verify that you have connectivity and the server is working.

During the installation and configuration of NSJMS, the following NSJMS location is created in the **nsjms.properties** file, for example:

```
Database.volsubvol=$DATA02.NSJMS
```

For additional information on using samples within eGate, see the *eGate Tutorial*

11.1.4. Implementing the HP NonStop JMS Message Service

1. Install and configure your NonStop JMS service. Refer to the HP NSJMS Installation documentation for instructions on how to do this or for additional information.
2. Set your shell variable to the following:

```
cd /usr/tandem/nsjms/<version>
export NSJMS_HOME=$PWD
export CLASSPATH=$NSJMS_HOME:$NSJMS_HOME/examples:$CLASSPATH
export CLASSPATH=`echo $NSJMS_HOME/lib/*.jar | tr ' ' ':'`:$CLASSPATH
```

3. Ensure you have a **nsjms.properties** file. If one does not exist, it may be copied from the **NSJMS_HOME** directory.
4. Ensure that the **Database.volsubvol** is set correctly. We suggest using the same **Database.volsubvol** as used by eGate.
5. Create a **jms.profile** to set **NSJMS_HOME** to the directory where the compiled samples are located. The sample programs expect to find the **nsjms.properties** file in the **$NSJMS_HOME** directory. It is a good practice to place a **nsjms.properties** file in the current directory.

Running the Sample Java Programs with NSJMS

Running the sample programs to interact with the NSJMSProj sample project.

1. Start the **SimpleReceiver** program to receive a message from Q1.

```
java SimpleReceiver -queue Q1
```

2. On a separate terminal, use the **SimpleSender** program to send 5 messages to Q2. The eGate API Kit sample project will receive messages from Q2 and send the message to Q1.

```
java SimpleSender -queue Q2 -count 5
```

When eGate is running you will see the following **SimpleReceiver**:

```
Rcv Message: eGate50-Message-Q2 Sending message 0
Rcv Message: eGate50-Message-Q2 Sending message 1
Rcv Message: eGate50-Message-Q2 Sending message 2
Rcv Message: eGate50-Message-Q2 Sending message 3
Rcv Message: eGate50-Message-Q2 Sending message 4
```

Illustrating that eGate is able to receive JMS messages from any Java program that sends NSJMS messages via the queue domain. It also illustrates that eGate can send

JMS message to any Java program that is to receive NSJMS message from the queue domain.

3 On a separate terminal, start the **SimpleConsumer** program to receive a message from T1.

```
java SimpleConsumer -topic T1
```

4 On a separate terminal use the **SimplePublisher** program to send 5 messages JMS topic T1. The eGate API Kit sample project will receive messages from T1 and send the message to T2

```
java -SimplePublisher -topic T1 -count 5
e.  When eGate is running you will see SimpleReceiver to show:
Rcv Message: eGate50-Message-T1 Publishing this message 0
Rcv Message: eGate50-Message-T1 Publishing this message 1
Rcv Message: eGate50-Message-T1 Publishing this message 2
Rcv Message: eGate50-Message-T1 Publishing this message 3
Rcv Message: eGate50-Message-T1 Publishing this message 4
```

This illustrates that eGate is able to receive JMS messages from any Java program that sends NSJMS messages via the topic domain. It also illustrates that eGate can send a JMS message to any Java program that is to receive NSJMS message from the queue domain.

11.1.5. Implementing the HP NonStop C/C++ Message Service

When using the C/C++ sample Makefiles, you will need to first install the NonStop JMS compiler NSJMS C++ API T2811V10_30SEP2003_V10. This compiler is available from Hewlett Packard.

1 Install and configure the NSJMS C++ API. During the installation process, the `/usr/tandem/nsjms/T2811V10_30SEP2003_V10` directory is created.

2 Copy the files located in the T2811V10_30SEP2003_V10 directory to a working directory. For example:

```
cp-Rp* <working-directory>/
```

3 Edit the Makefile **CAPI_HOME** to the following value:

```
cd <working-directory>
CAPI_HOME=/usr/tandem/nsjms/T2811V10_30SEP2003_V10
```

4 To build the four C sample programs, at a command prompt type “make” to create the SimpleConsumer, SimplePublisher, SimpleReceiver and SimpleSender executables.

5 Copy the **nsjms.properties** file from the NSJMS home directory to the working directory and if necessary edit it to match the nsjms.properties Database.volsubvol file used by eGate. If the file does not exist, you can copy it from the **NSJMS_HOME** directory.

6 Set your **Database.volsubvol** to the same **Database.volsubvol** that is used by eGate.

Set your **NSJMS_Home** to the working directory using a jms.profile. The test programs expect to find the nsjms.properties file in the **$NSJMS_HOME** directory. It is a good practice to use a **nsjms.properties** file in the current directory.

Running the Sample C/C++ Programs with NSJMS

1 Start the **SimpleReceiver** program to receive a message from a queue.

```
./SimpleRecevier -queue Q1 -count 10
```

2 On a separate terminal, use the **SimpleSender** program to send 5 messages to Q2. The eGate API Kit sample project will receive messages from Q2 and send the message to Q1.

```
./SimpleSender -queue Q2 -count 5
```

When eGate is running you will see SimpleReceiver show:

```
No message found, 3 sec delay ...
Rcv Message: eGate50-Message-Q2 Sending message 0
Rcv Message: eGate50-Message-Q2 Sending message 1
Rcv Message: eGate50-Message-Q2 Sending message 2
Rcv Message: eGate50-Message-Q2 Sending message 3
Rcv Message: eGate50-Message-Q2 Sending message 4
Rcv Message: eGate50-Message-Q2 Sending message 5
No message found, 3 sec delay ...
No message found, 3 sec delay ...
```

This illustrates that eGate is able to receive JMS messages from any C/C++ program that sends NSJMS messages via the queue domain. It also illustrates that eGate can send JMS message to any C/C++ program that is to receive NSJMS message from the queue domain.

3 On a separate terminal, start the **SimpleConsumer** to receive a message from T1.

```
java SimpleConsumer -topic T1
```

4 On a separate terminal use the SimplePublisher Java program to send 5 messages JMS topic T1. The eGate API Kit sample project will receive messages from T1 and send the message to T2

```
java -SimplePublisher -topic T1 -count 5
```

When eGate is running you will see SimpleReceiver show:

```
Rcv Message: eGate50-Message-T1 Publishing this message 0
Rcv Message: eGate50-Message-T1 Publishing this message 1
Rcv Message: eGate50-Message-T1 Publishing this message 2
Rcv Message: eGate50-Message-T1 Publishing this message 3
Rcv Message: eGate50-Message-T1 Publishing this message 4
```

This illustrates that eGate is able to receive JMS messages from any C/C++ program that sends NSJMS messages via the topic domain. It also illustrates that eGate can send JMS messages to any C/C++ program that is to receive NSJMS message from the queue domain.

11.2 Sample Code for Using NonStop JMS in Java

SeeBeyond JMS and NonStop JMS

The following illustrates the differences between NonStop JMS and SeeBeyond's JMS.

When using NonStop JMS you will need to replace STC Topic/Queue Factories with the following factories:

```
topicFactory: com.tandem.nsjms.client.jmpTopicConnecitonFactory
queueFactory: com.tandem.nsjms.client.jmsQueueConnectionFactory
```

After installing NonStop JMS APIs, sample code can be found in the **/usr/tandem/nsjms/T1251V20_30SEP2003_AAC/examples** directory.

11.3 Code for Using NonStop JMS in C/C++

SeeBeyond JMS and NonStop JMS

The following illustrates how to invoke NonStop JMS in C/C++ functions.

Topic subscriber

```
#include "nsjmsc.h"
....
char *propFile = getenv("NSJMS_HOME");
strcat(propFile, "/nsjms.properties");
pSes = new Session(propFile);

// or create a new session using a properties file in the current
working directory
//pSes = new Session("nsjms.properties");

// Creating the Consumer.
if (durable)
{
pConsumer = pSes->createDurableSubscriber(clientId, subName,
destName);
}
else
{
pConsumer = pSes->createConsumer(destName, Consumer::SUBSCRIBER);
}
 ....
```

Topic publisher

```
#include "nsjmsc.h"
....
char *propFile = getenv("NSJMS_HOME");
strcat(propFile, "/nsjms.properties");
pSes = new Session(propFile);

// or create a new session using a properties file in the current
working directory
//pSes = new Session("nsjms.properties");

// Creating the Producer.
pProducer = pSes->createProducer(destName);

....
```

Queue receiver

```
#include "nsjmsc.h"
....
char *propFile = getenv("NSJMS_HOME");
strcat(propFile, "/nsjms.properties");
pSes = new Session(propFile);
```

```
// or create a new session using a properties file in the current
working directory
//pSes = new Session("nsjms.properties");

// Creating the Consumer.
pConsumer = pSes->createConsumer(destName, Consumer::RECEIVER);
cout << "Receiver has been set" << endl;

long long start = 0;
for(int count = 0; count < msgCount; )
{
// Starting Transaction, Receiving the Message
BEGINTRANSACTION();
pMsg = pConsumer->receive(3000);
....
```

Queue Sender

```
#include "nsjmsc.h"
....
char *propFile = getenv("NSJMS_HOME");
strcat(propFile, "/nsjms.properties");
pSes = new Session(propFile);

// or create a new session using a properties file in the current
working directory
//pSes = new Session("nsjms.properties");

// Creating the Producer.
pProducer = pSes->createProducer(destName);

long long start = JULIANTIMESTAMP();
for (unsigned long count = 0; count < msgCount; count++)
{
strstream msg;

// Setting Message
msg<<"Sending message "<<count<<ends;
cout << msg.str() << endl;

// Creating Text Message
pMsg = pSes->createTextMessage();
pMsg->setStringProperty("test","SimpleSender");

// Setting Text
pMsg->setText (msg.str());

pProducer->send(pMsg, deliveryMode, PRIORITY, DEFAULT_JMSEXPIRATION);

....
```

Index

A

B

C

D

E

G

H

I

J

L

M

N

O

P

Q

R

S

T

U

W

X