SUN SEEBEYOND

eGATE™ API KIT FOR JMS IQ MANAGER (COM+ EDITION)

Release 5.1.1

Sun microsystems

Version 20060627133157

Contents

Chapter 1

Introduction

This chapter introduces you to this guide, its general purpose and scope, and its organization. It also provides sources of related documentation and information.

What's in This Chapter

1.1 About This Document

This user's guide describes how to install and use the eGate™ API Kit to create COM+ applications that connect to Sun Java™ Composite Platform Suite (CAPS) Projects via Java™ Message Service (JMS).

1.1.1 What's in This DocumentTopic

This document includes the following chapters:

- **Chapter 2 "Installing the eGate API Kit" on page 17** describes how to install the eGate API Kit and its samples.
- **Chapter 3 "JMS and COM+ Implementation Overview" on page 20** gives information about the JMS IQ Manager and how the COM+ API interfaces with it.
- **Chapter 4 "COM+ Object Reference" on page 40** describes how to develop COM+ applications to access the Sun SeeBeyond JMS IQ Manager in Java CAPS Project.
- **Chapter 5 "Working with the COM+ API Samples" on page 120** describes the COM+ samples and how to configure and implement them.

1.1.2 Intended Audience

This guide is intended for developers who are familiar with programming applications that interface through JMS.

1.1.3 Text Conventions

The following conventions are observed throughout this document.

Table 1 Text Conventions

Text Convention	Used For	Examples
Bold	Names of buttons, files, icons, parameters, variables, methods, menus, and objects	▪ Click **OK**. ▪ On the **File** menu, click **Exit**. ▪ Select the **eGate.sar** file.
`Monospaced`	Command line arguments, code samples; variables are shown in ***bold italic***	`java -jar` ***`filename`***`.jar`
Blue bold	Hypertext links within document	See **Related Documents** on page 15
Blue underlined	Hypertext links for Web addresses (URLs) or email addresses	http://www.sun.com

1.1.4 Screenshots

Depending on what products you have installed, and how they are configured, the screenshots in this document may differ from what you see on your system.

1.2 Related Documents

For more information about eGate Integrator, refer to the following documents:

- *Sun SeeBeyond Java Composite Application Platform Suite Installation Guide*
- *Sun SeeBeyond eGate Integrator User's Guide*
- *Sun SeeBeyond eGate Integrator JMS Reference Guide*
- *Sun SeeBeyond eGate Integrator System Administrator Guide*
- *Sun SeeBeyond Java Composite Application Platform Suite Deployment Guide*

1.3 Sun Microsystems, Inc. Web Site

The Sun Microsystems web site is your best source for up-to-the-minute product news and technical support information. The site's URL is:

http://www.sun.com

1.4 Documentation Feedback

We appreciate your feedback. Please send any comments or suggestions regarding this document to:

CAPS_docsfeedback@sun.com

Chapter 2

Installing the eGate API Kit

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

What's in This Chapter

2.1 Supported Operating Systems

For information about supported operating systems, refer to the **eGateAPIKit_Readme.txt**.

2.2 System Requirements

The eGate API Kit has the same system requirements as eGate Integrator. For information, refer to the *Sun SeeBeyond Java Composite Application Platform Suite Installation Guide*.

In addition, you need a development environment with a compiler that is compatible with the selected O/S; for example, Microsoft Visual .Net 2003.

2.3 Supported Compilers

The list below provides the compilers that you can use to compile your COM+ application. If you use a different compiler, be aware that some compilers are incompatible with eGate.

Table 2 Supported Compilers

Operating System	Compiler
Windows	.Net 2003 version 7.1
IBM AIX	Visual Age for C++ 6.0, 64 bit
IBM AIX	Visual Age for C++ 6.0, 32bit
Sun Solaris	Sun ONE Studio 7
HP-UX	aCC 3.37, 64 bit
HP Itanium	aCC 5.36, 64 bit
Red Hat Enterprise AS 3	Advanced Server 3
Tru64	Tru64 5.1A and 5.1B

2.4 Installing the eGate API Kit

The procedure below describes an overview of how to install Sun SeeBeyond eGate API Kit. For detailed installation instructions, refer to the *Sun Java SeeBeyond Composite Application Platform Suite Installation Guide.*

Before you install the Sun SeeBeyond eGate API Kit, install and download the following items using the Java CAPS Installer:

- Repository
- eGate Integrator
- Enterprise Designer
- Enterprise Manager
- Logical Host

The procedure below describes how to install the following items for Sun SeeBeyond eGate API Kit:

- the software
- the documentation
- the sample Enterprise Designer Projects and the code samples

To install Sun SeeBeyond eGate API Kit

1. Launch the Java Composite Application Platform Suite Installer.
2. In the **Administrator** page, click **Click to install additional products**.
3. In the list of products to install, select the following:
   - **eGate API Kit > eGate_APIKit_<OS>.sar** where <OS> is the operating system you are installing on (to install the Sun SeeBeyond eGate API Kit software)

- **Documentation > eGateAPIKitDocs** (optional—to install the Sun SeeBeyond eGate API Kit documentation and samples)

4. In the **Administrator > Upload** page, select the following items and click **Next** after each SAR file is selected:
   - **eGate_APIKit_<OS>.sar** (for example, **eGate_APIKit_SunOS.sar**).
   - **eGateAPIKitDocs.sar**

   When the installation is finished, the "Installation Completed" message appears.
5. In the **Downloads** page, select **API kit for <OS>** (where <OS> is the operating system you are installing on), select a location for the **.zip** file to be saved, and then extract the file.
6. To download the sample Projects and code samples, click **Download Sample**, and select a location for the **.zip** file to be saved.

   For information about importing and using the sample Projects, refer to **Chapter 5 "Working with the COM+ API Samples"**.

2.5 Post-Installation Instructions

After the eGate API Kit installation, do the following before you start building applications:

1. Locate the **stc_mscom.dll**, **stc_msclient.dll**, **stc_mscommon.dll**, **stc_msapi.dll** files in the directory on the external system where the eGate API Kit is installed.
2. From the command line of the external system, register the file **stc_mscom.dll** into the Windows Registry as illustrated below:

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

Chapter 3

JMS and COM+ Implementation Overview

The eGate API Kit provides an interface for external applications to exchange data with Java CAPS Projects via the Sun SeeBeyond JMS IQ Manager. This chapter gives an overview of JMS and the Sun SeeBeyond JMS IQ Manager, and provides implementation and CRM information.

What's in This Chapter

3.1 About the Sun SeeBeyond JMS IQ Manager

This section provides an overview of the Sun SeeBeyond JMS IQ manager, including JMS version support and considerations for the Java CAPS Project. This section also describes how to find the port numbers used for a particular runtime Project.

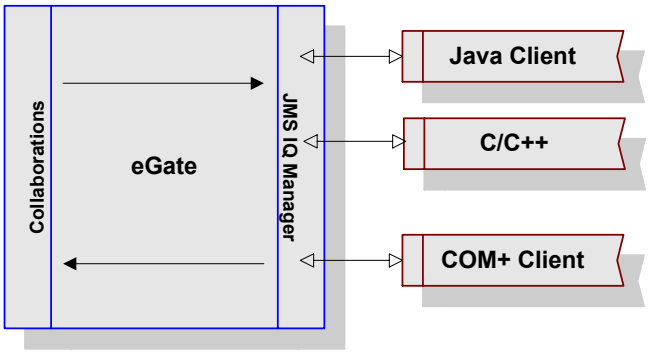
3.1.1 The Java CAPS JMS Interface

For those of you unfamiliar with JMS interfaces, this section describes the Java CAPS JMS interface. The Java CAPS JMS consists of the following components:

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

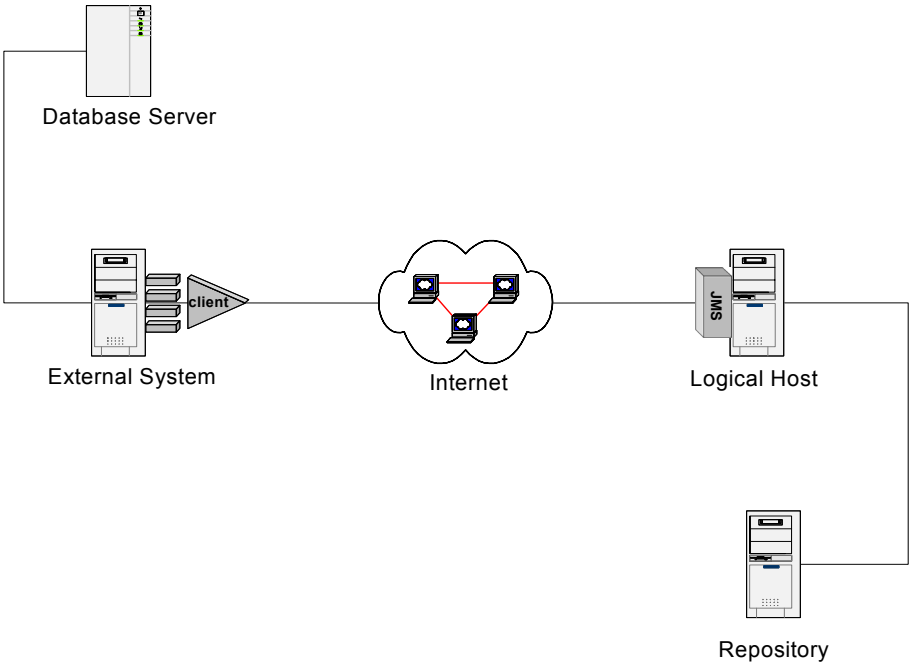
Figure 1 illustrates the communication between each component.

Figure 1 Message Service Communication Architecture



In Figure 2, all necessary components have been isolated onto a separate system. While this separation is not mandatory, the combinations of components that reside together on various systems, change depending upon your needs.

Figure 2 Java CAPS TCP/IP Communication Architecture



In some form, the following components must exist:

- Repository
- Logical Host
- 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 Logical Host and the External System may exist on one physical machine.

3.1.2 Java CAPS Project Considerations

To enable your application to communicate with a runtime JMS IQ Manager, consider the following:

- The message destination names and the names of the components used must coincide.
- Your JMS application must use the expected data format, the name of the message destination, the name of host and port number of the JMS IQ Manager (see **"About the Sun SeeBeyond JMS IQ Manager"** for port number information).
- The methods used must correspond to the expected data format.

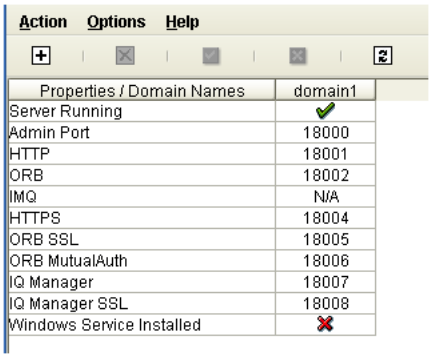
3.1.3 Viewing JMS IQ Manager Port Numbers

The default port number for JMS IQ Managers is 18007. The default port number for SSL is 18008. To view the port numbers for your runtime Java CAPS Project, use the Domain Manager as described in the procedure below.

To view JMS IQ Manager port numbers

1. Navigate to the folder where the Java CAPS Logical Host is installed.
2. Double-click **domainmgr.bat**. The Domain Manager window appears.

Figure 3 Viewing Runtime JMS IQ Manager Port Numbers



The **IQ Manager** field shows the JMS IQ Manager port; the **IQ Manager SSL** field shows the JMS IQ Manager SSL port.

3.2 About the Java Messaging Service

This section provides an overview of JMS messages and some different types of messaging scenarios.

3.2.1 JMS Messages

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

- **Header** - The header fields contain values used by both clients and providers to identify and route messages. All messages support the same set of header fields.
- **Properties** - The properties provide a way to add optional header fields to messages. They can be application-specific, standard, or provider-specific.
- **Body** (or **Payload**) - JMS supports 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. The fields in the header are described below.

- **JMSDestination** - The destination to which the message is being sent.
- **JMSDeliveryMode** - 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** - A value that uniquely identifies 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 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 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** - An indicator of whether the message was re-delivered to the consumer. If the header is "true", the message is re-delivered; If the header is false, the message is not. The message might 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 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** - The javax.jms.Destination, which indicates the address to which to reply and enables the consumer to reply to a message associated with a specific producer.

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

- **JMSCorrelationID** - Associates 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, and is not limited to JMSMessageID.

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

Message Properties

Properties allow a client to have the JMS provider select messages based on application-specific criteria using message selectors. 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 the eGate API Kit:

- **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.

3.2.2 JMS Messaging Types

This section discusses characteristics of the following types of messaging scenarios.

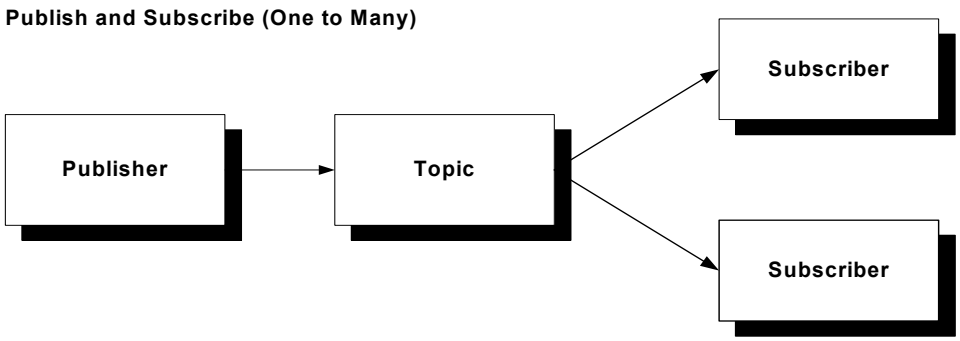
- **Publish/Subscribe Projects** on page 24
- **Point-to-Point Projects** on page 25
- **Request-Reply Projects** on page 26

Publish/Subscribe Projects

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 requiring a request. They are pushed via a channel referred to as a topic. The 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 4 The Publish/Subscribe Schema

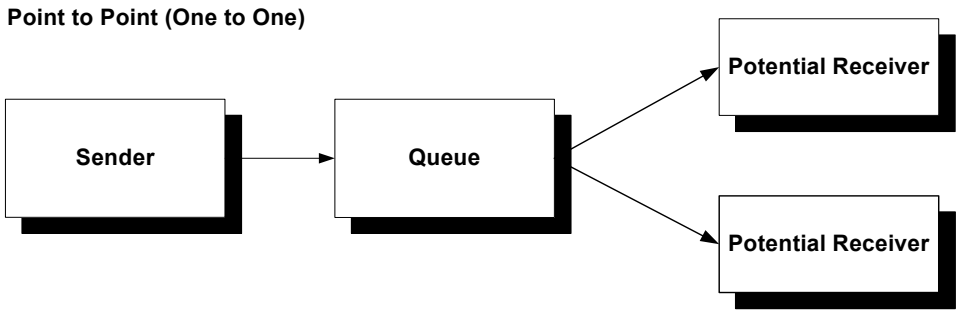


Point-to-Point Projects

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 instead of a topic. 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.

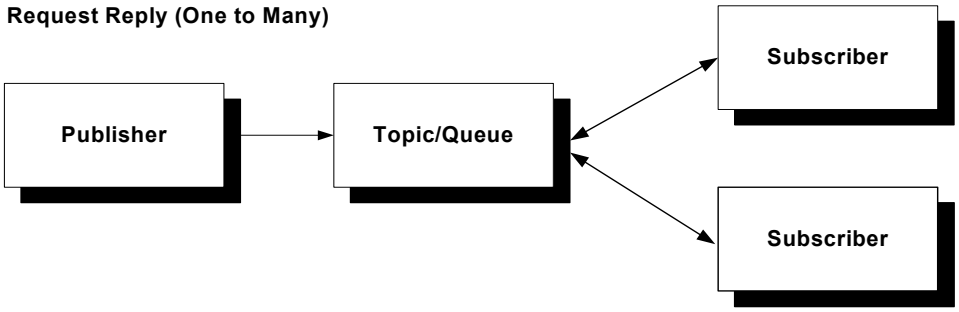
Figure 5 Point to Point



Request-Reply Projects

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 6 The Request-Reply Schema



Following is a scenario that provides an example of how a request-reply project could be configured.

1. A request is received by the **JMS Connection,** which is controlled by the JMS IQ Manager, and 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.

3.3 About the COM+ Interface

The COM+ Edition of the eGate API Kit supports the *Java Messaging Service Specification version 1.0.2b*. This section provides overview information for the following topics:

- **Creating Destinations** on page 27
- **XA Compliance** on page 27
- **Message Selectors** on page 27
- **Multi-Threaded Apartments** on page 27
- **Sample Code** on page 28

3.3.1 Creating Destinations

Destinations do not need to be created separately; they are created through the `QueueSession.CreateQueue()` and `QueueSession.CreateTopic()` functions. If these destinations do not exist, they are created automatically.

3.3.2 XA Compliance

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. This is known as a transactional environment.

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

3.3.3 Message Selectors

A message selector allows a client to specify the messages in which the client is interested using the message header. Only messages for which the headers and properties match the selector are delivered. The semantics of not delivered differ depending on the message consumer 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 the *Java Messaging Service Specification version 1.0.2b.*

3.3.4 Multi-Threaded Apartments

The COM+ components support the multi-threaded apartment model (MTA). Multiple threads in the application can use the COM+ component at the same time; for example, 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 *Java Messaging Service Specification version 1.0.2b*. In essence, this defines the session and its associated objects as single threaded contexts.

For the JMS API in COM+, all the components were built using Microsoft Visual Studio .Net 2003 with the msvcp71.dll and the msvcr71.dll libraries. Applications that use the COM+ components should be compiled with the runtime libraries option: "multi-threaded DLL" to assure the same runtime libraries are used.

3.3.5 Sample Code

The eGate API Toolkit provides a sample download that includes code samples for creating interfaces to the Java CAPS JMS with COM+. For information about implementing the sample code, see **"Working with the COM+ API Samples" on page 120**.

3.4 The Compensating Resource Manager (CRM)

This section provides basic information about the CRM and also provides instructions for configuring the CRM for the eGate API Toolkit. It includes the following topics:

3.4.1 About Compensating Resource Managers

A *Compensating Resource Manager* can be described as a COM+ object that uses a set of tools (CRM facility) that allow you to create resource managers. This allows you 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

Implementing distributed transactions is the key to the two-phase commit protocol. 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.

3.4.2 Implementing the Compensating Resource Manager

When planning a 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 the power source is inadvertently disconnected 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 Component Services administrative tool. The CRM Worker and CRM Compensator must be installed into the same COM+ application. For information about configuring and running the CRM samples, see **"Building the CRM Sample Application" on page 124**.

3.4.3 Configuring the Compensating Resource Manager

To enable the CRM functionality, the Component Services must be configured using the Windows Component Services administrative tool as described in the following steps.

Note: *Before beginning the following procedure, make sure the stc_mscom.dll file is registered, as described in* **"Post-Installation Instructions" on page 19**.

Step 1: Add a New Component Application

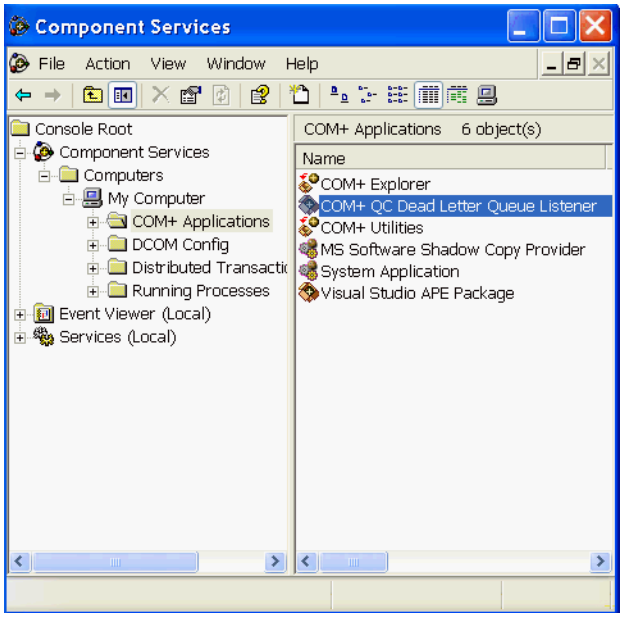
This procedure guides you through using the COM+ Application Install Wizard to create a new Component Application.

To add a new Component Application

1 Open the Component Services window from the Control Panel (select **Administrative Tools** and then select **Component Services**).

2 Expand the **Component Services** folder (see Figure 7) and right-click **COM+ Applications**.

Figure 7 Component Services Folder



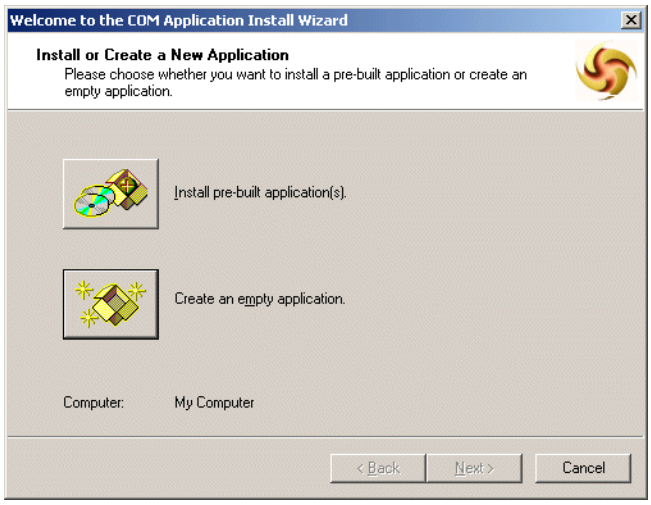
3 On the context menu, click **New**, and then click **Application**. The COM+ Application Install Wizard opens (see Figure 8).

Figure 8 COM+ Application Install Wizard

4 Click **Next** to continue.

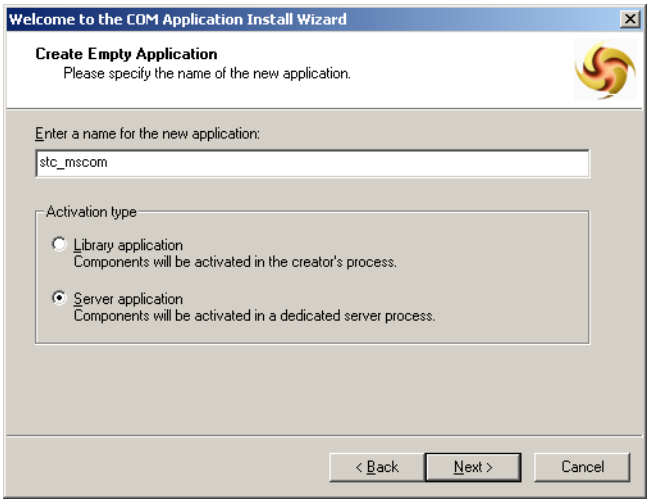
5 On the **Install or Create a New Application** window, click **Create an empty application** (see Figure 9).

Figure 9 COM+ Application Install Wizard



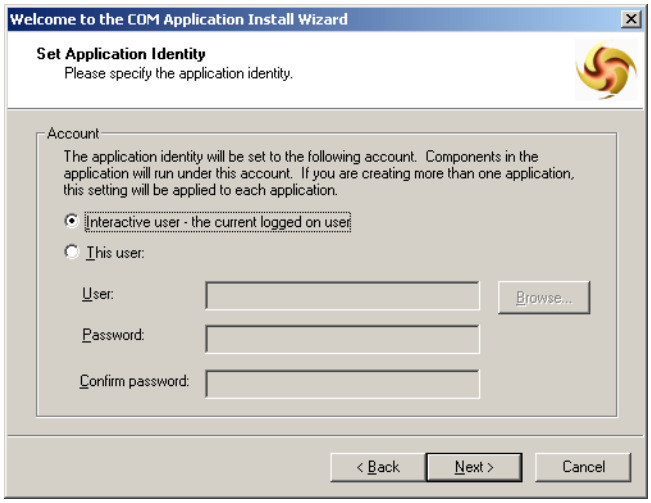
6 On the **Create Empty Application** window, enter the name **stc_mscom**, click the option button next to **Server application**, and then click **Next**.

Figure 10 COM+ Application Install Wizard: New Application



7 On the **Set Application Identity** page, click **Interactive User**, and then click **Next**.

Figure 11 COM+ Application Install Wizard: Set Application Identity



8 Click **Finish**.

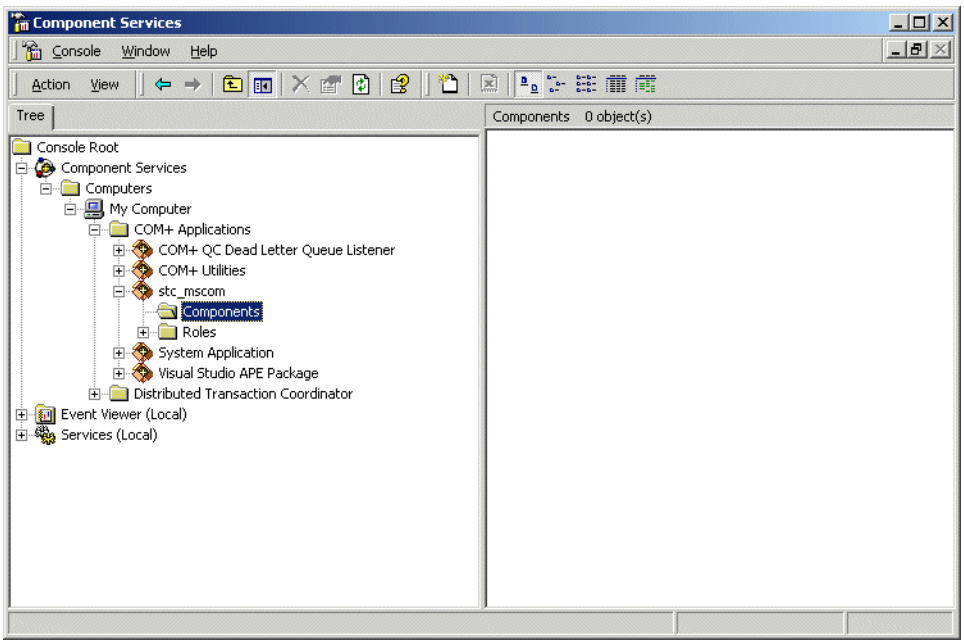
Step 2: Installing the stc_mscom Component

Once you create the Component Application, you can install the eGate API Kit library files for COM+.

To install the stc_mscom component

1 From the Component Services window of the Control Panel, expand the **stc_mscom** component under **COM+ Applications** (see Figure 12).

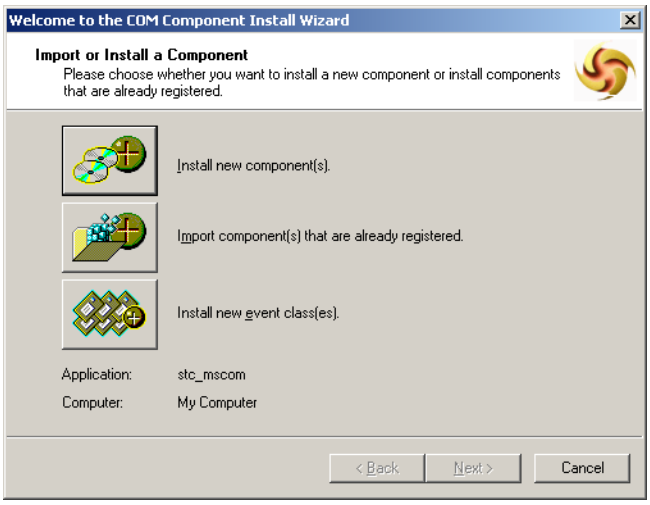
Figure 12 Component Services: stc_mscom Component



2 Right-click the **Components** folder. On the shortcut menu, click **New**, and then click **Component**. The COM+ Component Install Wizard appears.

3 On the COM+ Component Install Wizard, click **Next**. The **Import or Install a Component** window appears.

Figure 13 COM+ Component Install Wizard

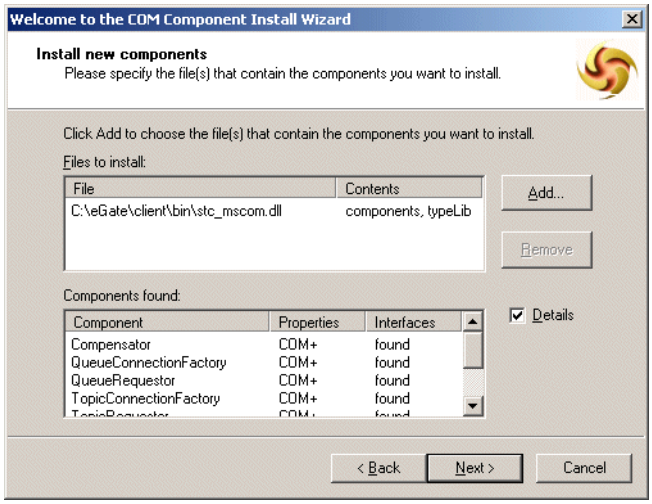


4 Click **Install new component(s)**. The **Select files to install** dialog appears.

5 In the **Select files to install** dialog, locate and select the file **stc_mscom.dll**, and then click **Open**. The **Install new components** window appears.

Note: *The **stc_mscom.dll** file is located in the directory where you extracted the eGate API Kit when you installed the application (see* **"Installing the eGate API Kit" on page 18***).*

Figure 14 COM+ Component Install Wizard: Add



6 On the **Install new components** window, select the **Details** check box next to the **Components found** pane, and then click **Next** to continue.

7 Click **Finish**.

Step 3: Configuring the stc_mscom Component

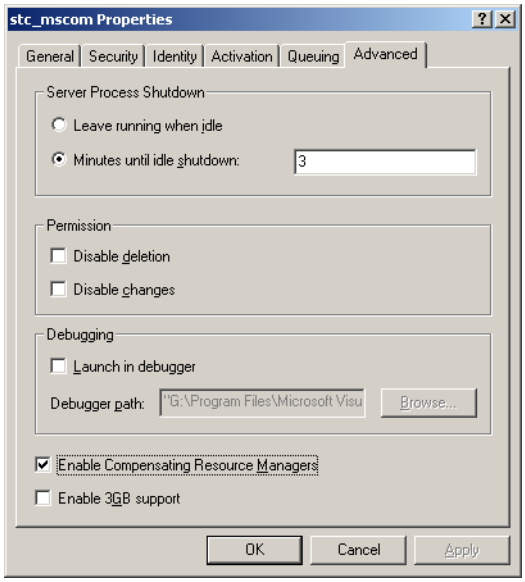
Once you install the eGate API Kit library files, you must configure them to enable CRM.

To configure the stc_mscom component

1 Right-click the **stc_mscom** component and, on the shortcut menu, click **Properties**. The **stc_mscom Properties** dialog appears.

2 Click the Advanced tab, select the **Enable compensating resource managers** check box, and then click **OK**.

Figure 15 stc_mscom Properties: Advanced



Step 4: Configuring the STC_MSCOM.Compensator

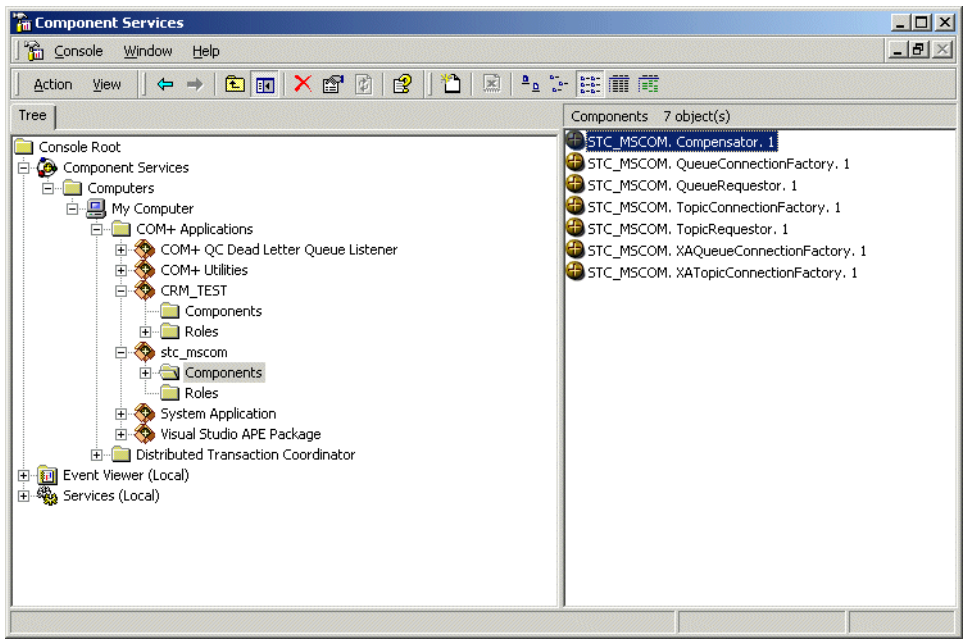
After you configure the eGate API Kit library files, you need to configure certain components, beginning with the compensator.

To configure the STC_MSCOM.Compensator

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

2 In the **Components** pane on the right side of the window (see Figure 16), right-click **STC_MSCOM.Compensator**, and then click **Properties**.

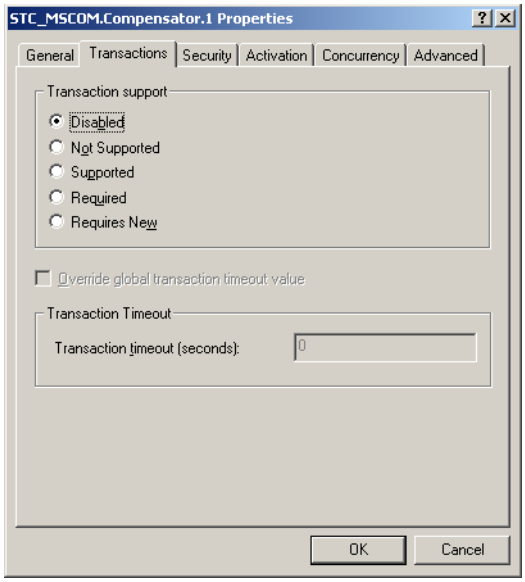
Figure 16 STC_MSCOM.Compensator Properties



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

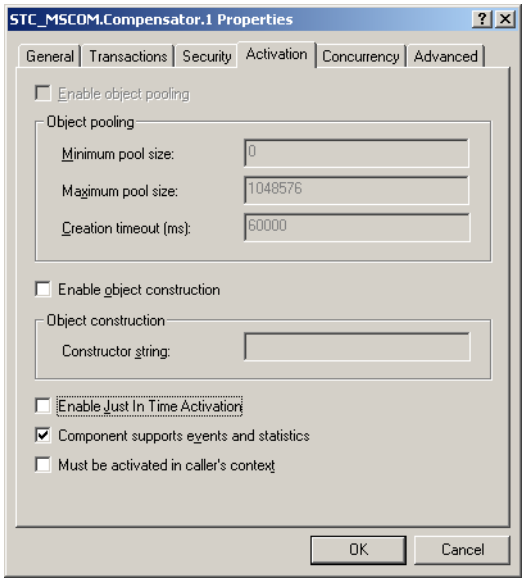
3 Click the Transactions tab, and then select **Disabled** in the **Transaction support** pane.

Figure 17 STC_MSCOM.Compensator Properties: Transaction Support



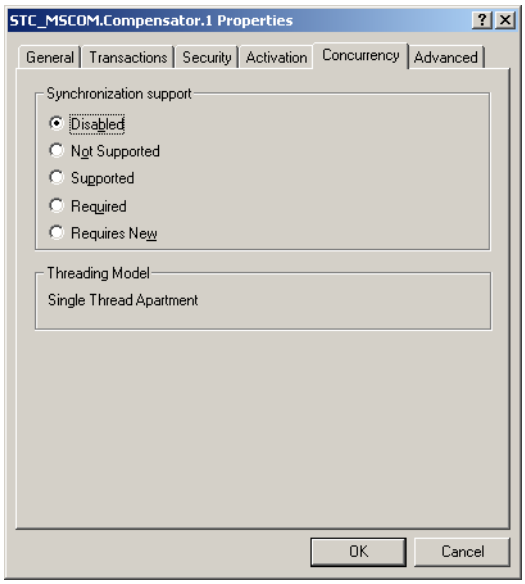
4 Click the Activation tab, and then deselect the **Enable Just In Time Activation** check box.

Figure 18 STC_MSCOM.Compenstator Properties: Activation



5 Click the Concurrency tab, and then select **Disabled** for **Synchronization Support**.

Figure 19 STC_MSCOM.Compensator Properties: Concurrency



6 Click **OK**.

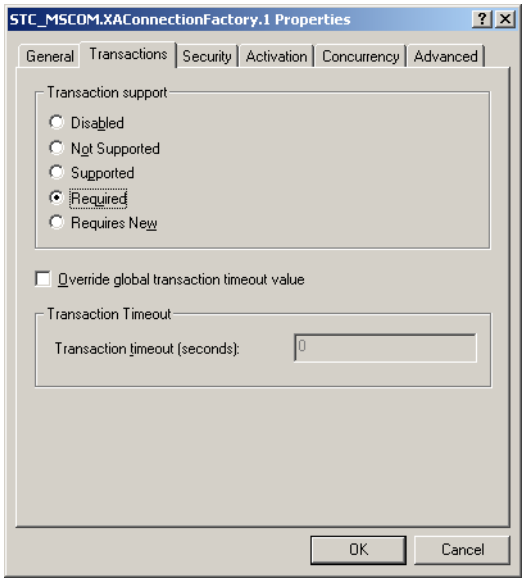
Step 5: Configuring the STC_MSCOM XA Connection Factories

The final step in configuring the eGate API Kit library files is to configure to XAQueueConnectionFactory component.

To configure the STC_MSCOM XA Connection Factories

1. In the **Components** pane, right-click **STC_MSCOM.XAQueueConnectionFactory** and then click **Properties**.
2. Click the Transactions tab, and then select **Required** for **Transaction support**.

Figure 20 STC_MSCOM.XAConnectionFactory Properties: Transaction Support



3. Click the Activation tab, and select **Enable Just In Time Activation** (this option should be already selected and might be disabled).
4. Click the Concurrency tab, and select **Required** for **Synchronization support** (this should be the only selectable option), and then click **OK**.
5. Repeat the above steps for the **STC_MSCOM.XATopicConnectionFactory** component.

Chapter 4

COM+ Object Reference

The eGate API Kit provides an interface for COM+ applications to exchange data with Java CAPS Projects via the Sun SeeBeyond JMS IQ Manager. This chapter describes how to implement the Java CAPS JMS API in COM+ applications.

What's in This Chapter

4.1 Supported JMS Classes for COM+

eGate supports the Java Message Service (JMS) COM+ APIs listed in the following sections. As much as possible, the COM+ API was designed to correspond to the standard JMS Java API. If you need additional information for each of the classes and methods, please refer to the Sun Microsystems web site at:

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

This documentation provides a reference to the Java Message Service API, and can help you understand how the COM+ methods are used in a JMS implementation.

You may also find the following books useful:

- *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

4.2 Viewing the COM+ API

You can view the JMS COM+ APIs using any application that is capable of viewing COM+ APIs. To view the APIs using Microsoft Visual Studio .NET, open any Visual Basic project file(.vbp) in the COM+ samples (for information about locating the sample files, see **"About the COM+ Samples" on page 120**).

To begin viewing the APIs

1. Start Microsoft Visual Basic 6.0.
2. In the **New Project** dialog box, click **Standard EXE** and then click **Open**.
3. On the **Project** toolbar, click **References**.
4. In the **References** dialog box, select **Sun Java Composite Application Platform Suite Message Service 5.1.0**, and then click **OK**.
5. On the **View** toolbar, click **Object Browser**.
6. From the **All Libraries** list box, select **STC_MSCOM**.
7. Press the F2 button to open the **Object Browse**r dialog box.
8. From the **All Libraries** drop-down button, select **STC_MSCOM** to view the supported classes and methods.
9. Highlight the class to view the member methods and properties.

Note: *You can also view the API in .NET by opening any of the sample projects and then viewing* ***stc_mscom.dll*** *in the Object Viewer.*

4.3 COM+ Constants

The COM+ API for SeeBeyond JMS defines values for the following types of constants:

- **AcknowledgeMode Constants** on page 42
- **DeliveryMode Constants** on page 43
- **Message Constants** on page 43
- **Priority Constants** on page 44

For a list of error code constants, see **"Error Value Constants" on page 119**.

4.3.1 AcknowledgeMode Constants

Table 3 Values for AcknowledgeMode Constants

Name	Value	Description
msAutoAcknowledge	1	**1** indicates 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.

Table 3 Values for AcknowledgeMode Constants

Name	Value	Description
msClientAcknowledge	2	**2** indicates 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.
msDupsOkAcknowledge	3	**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.

4.3.2 DeliveryMode Constants

Table 4 Values for DeliveryMode Constants

Name	Value	Description
msNonPersistent	0	**0** indicates non-persistent delivery mode. This mode maximizes performance, and should be used if an occasional lost message is tolerable.
msPersistent	1	**1** indicates persistent delivery mode. This mode maximizes reliability, and should be used if the application will have problems if the message is lost in transit.

4.3.3 Message Constants

Table 5 Values for Message Constants

Name	Value	Description
msDefaultDeliveryMode	1	See **"DeliveryMode Constants" on page 43**.
msDefaultPriority	4	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.
msDefaultTimeToLive	0	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.

4.3.4 Priority Constants

Table 6 Values for Priority Constants

Name	Value
msPriorityEight	8
msPriorityFive	5
msPriorityFour	4
msPriorityNine	9
msPriorityOne	1
msPrioritySeven	7
msPrioritySix	6
msPriorityThree	3
msPriorityTwo	2
msPriorityZero	0

Priority constants define a ten-level priority system for messages. See **"msDefaultPriority" on page 43**.

4.4 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. The BytesMessage Object is a member of the Message Object.

4.4.1 BytesMessage Object Methods

The BytesMessage object includes the following methods:

Acknowledge Method

Acknowledges the receipt of current and previous messages.

```
BytesMessage.acknowledge
```

ClearBody Method

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

```
BytesMessage.ClearBody
```

ClearProperties Method

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

```
BytesMessage.ClearProperties
```

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.

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.

ReadBoolean Method

Reads a Boolean value from the bytes message stream.

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

ReadByte Method

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

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

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.

ReadChar Method

Reads a Unicode character value from the bytes message stream.

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

ReadDouble Method

Reads a double from the bytes message stream.

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

ReadFloat Method

Reads a float from the bytes message stream.

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

ReadInt Method

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

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

ReadLong Method

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

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

ReadShort Method

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

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

ReadUnsignedByte Method

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

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

ReadUnsignedShort Method

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

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

ReadUTF Method

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

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

Reset Method

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

```
BytesMessage.Reset
```

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.

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.

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

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.

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.

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.

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.

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.

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.

WriteObject Method

Currently not supported

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.

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.

4.4.2 BytesMessage Object Properties

The BytesMessage object includes the following properties:

- **CorrelationID Property** on page 50
- **CorrelationIDAsBytes Property** on page 50
- **DeliveryMode Property** on page 50
- **Destination Property** on page 51
- **Expiration Property** on page 51
- **MessageID Property** on page 51
- **Priority Property** on page 51
- **Redelivered Property** on page 51
- **ReplyTo Property** on page 51
- **Timestamp Property** on page 52
- **Type Property** on page 52

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
```

CorrelationIDAsBytes Property

Currently not supported.

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.

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.

Destination Property

Currently not supported.

Expiration Property

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

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

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
```

Priority Property

Currently not supported.

Redelivered Property

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

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

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
```

Timestamp Property

The TimeStamp property sets or returns the message timestamp.

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

Type Property

The Type property sets or returns the message type.

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

4.5 Connection Object

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

4.5.1 Connection Object Methods

The Connection object includes the following methods:

- **Start Method** on page 52
- **Stop Method** on page 52

Start Method

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

```
Connection.Start
```

Stop Method

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

```
Connection.Stop
```

4.5.2 Connection Object Properties

The Connection object includes the following properties:

- **ClientID Property** on page 53
- **MetaData Property** on page 53

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
```

MetaData Property

This property is not currently supported.

4.6 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.

4.6.1 ConnectionFactory Object Properties

The ConnectionFactory object includes the following properties:

- **HostName Property** on page 53
- **Port Property** on page 53
- **PortOffset Property** on page 54

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
```

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
```

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
```

4.7 Connection MetaData Object

This Object is currently not supported.

4.8 MapMessage Object

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

4.8.1 MapMessage Object Methods

The MapMessage object includes the following methods:

Acknowledge Method

Acknowledges the receipt of current and previous messages.

```
MapMessage.Acknowledge
```

ClearBody Method

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

```
MapMessage.ClearBody
```

ClearProperties Method

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

```
MapMessage.ClearProperties
```

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.

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.

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

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.

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.

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.

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.

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.

GetObject Method

The GetObject method is currently not supported.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

SetObject Method

This method is currently not supported.

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.

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.

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.

4.8.2 MapMessage Object Properties

The MapMessage object includes the following properties:

- **CorrelationID Property** on page 61
- **CorrelationIDAsBytes Property** on page 61
- **DeliveryMode Property** on page 61
- **Destination Property** on page 61
- **Expiration Property** on page 61
- **MessageID Property** on page 62
- **Priority Property** on page 62
- **Redelivered Property** on page 62
- **ReplyTo Property** on page 62
- **Timestamp Property** on page 62

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
```

CorrelationIDAsBytes Property

```
Currently not supported.
```

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.

Destination Property

Currently not supported.

Expiration Property

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

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

MapNames Property

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

```
MapMessage.MapNames = Variant
```

```
Variant = MapMessage.MapNames
```

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
```

Priority Property

Currently not supported.

Redelivered Property

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

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

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
```

Timestamp Property

The TimeStamp property sets or returns the message timestamp.

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

Type Property

The Type property sets or returns the message type.

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

4.9 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.

4.9.1 Message Object Methods

The Message object includes the following methods:

Acknowledge Method

Acknowledges the receipt of current and previous messages.

```
Message.acknowledge
```

ClearBody Method

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

```
Message.ClearBody
```

ClearProperties Method

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

```
Message.ClearProperties
```

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.

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.

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.

4.9.2 Message Object Properties

The Message object includes the following properties:

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
```

CorrelationIDAsBytes Property

The CorrelationIDAsBytes is not currently supported.

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.

Destination Property

```
Currently not supported.
```

Expiration Property

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

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

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
```

Priority Property

Currently not supported.

Redelivered Property

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

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

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
```

Timestamp Property

The TimeStamp property sets or returns the message timestamp.

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

Type Property

The Type property sets or returns the message type.

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

4.10 MessageConsumer Object

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

4.10.1 MessageConsumer Object Methods

The MessageConsumer object includes the following methods:

- **Close Method** on page 66
- **Receive Message Method** on page 67
- **ReceiveNoWait Method** on page 67

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
```

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.

ReceiveNoWait Method

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

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

4.10.2 MessageConsumer Object Properties

The MessageConsumer object includes the following properties:

- **MessageListener Property** on page 67
- **MessageSelector Property** on page 67

MessageListener Property

This property is currently not supported.

MessageSelector Property

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

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

4.11 MessageListener Object

This object is currently not supported.

4.12 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.

4.12.1 MessageProducer Object Properties

The MessageProducer object includes the following properties:

- **DeliveryMode Property** on page 68
- **DisableMessageID Property** on page 68
- **DisableMessageTimestamp Property** on page 68

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.

DisableMessageID Property

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

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

DisableMessageTimestamp Property

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

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

Priority Method

Currently not supported.

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
```

4.13 Queue Object

A Queue object encapsulates a Message Service specific queue name.

4.13.1 Queue Object Methods

The Queue object includes the following method:

- **ToString Method** on page 69

ToString Method

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

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

4.13.2 Queue Object Properties

The Queue object includes the following property:

- **QueueName Property** on page 69

QueueName Property

Returns the name of this queue. Read-only.

4.14 QueueBrowser Object

This object is currently not supported.

4.15 QueueConnection Object

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

4.15.1 QueueConnection Object Methods

The QueueConnection object includes the following methods:

- **CreateQueueSession Method** on page 70
- **Start Method** on page 70
- **Stop Method** on page 70

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.
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.

Start Method

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

```
QueueConnection.Start
```

Stop Method

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

```
QueueConnection.Stop
```

4.15.2 QueueConnection Object Properties

The QueueConnection object includes the following properties:

- **ClientID Property** on page 71
- **MetaData Property** on page 71

ClientID Property

Returns or sets client identifier for this connection.

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

MetaData Property

Not currently supported.

4.16 QueueConnectionFactory Object

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

4.16.1 QueueConnectionFactory Object Methods

The QueueConnectionFactory object includes the following method:

- **CreateQueueConnection Method** on page 71

CreateQueueConnection Method

Create a queue connection with a default user identity.

```
QueueConnectionFactory.CreateQueueConnection() As QueueConnection

QueueConnectionFactory.CreateQueueConnection(String user, String
password) As QueueConnection
```

4.16.2 QueueConnectionFactory Object Properties

The QueueConnectionFactory object includes the following properties:

- **HostName Property** on page 72
- **Port Property** on page 72
- **PortOffset Property** on page 72

HostName Property

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

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

Port Property

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

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

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
```

4.17 QueueReceiver Object

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

4.17.1 QueueReceiver Object Methods

The QueueReceiver object includes the following methods:

- **Close Method** on page 72
- **Receive Method** on page 73
- **ReceiveNoWait Method** on page 73

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
```

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.

ReceiveNoWait Method

Receive the next message if one is immediately available.

```
QueueReceiver.ReceiveNoWait As message
```

4.17.2 QueueReceiver Object Properties

The QueueReceiver object includes the following properties:

- **MessageListener Property** on page 73
- **MessageSelector Property** on page 73
- **Queue Property** on page 73

MessageListener Property

This property is not currently supported.

MessageSelector Property

Returns this message consumer's message selector expression.

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

Queue Property

Returns the queue associated with this queue receiver.

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

4.18 QueueRequestor Object

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

4.18.1 QueueRequestor Object Methods

The QueueRequestor object includes the following methods:

- **Create Method** on page 74
- **Request Method** on page 74

Create Method

Constructs the QueueRequestor.

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

Name	Description
session	The QueueSession.
queue	Queue name.

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.

4.19 QueueSender Object

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

4.19.1 QueueSender Object Methods

The QueueSender object includes the following method:

- **Send Method** on page 74

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.

4.19.2 QueueSender Object Properties

The QueueSender object includes the following properties:

- **DeliveryMode Property** on page 75
- **DisableMessageID Property** on page 75
- **DisableMessageTimestamp Property** on page 76
- **Priority Property** on page 76
- **Queue Property** on page 76
- **TimeToLive Property** on page 76

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.

DisableMessageID Property

Returns or sets an indication of whether message IDs are disabled

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

DisableMessageTimestamp Property

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

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

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.

Queue Property

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

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

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
```

4.20 QueueSession Object

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

4.20.1 QueueSession Object Methods

The QueueSession object includes the following methods:

- **CreateReceiver Method** on page 78
- **Run Method** on page 79

Commit Method

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

```
QueueSession.Commit
```

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.

CreateBytesMessage Method

Create a BytesMessage.

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

CreateMapMessage Method

Create a MapMessage.

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

CreateMessage Method

Create a Message.

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

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.

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.

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.

CreateStreamMessage Method

Create a StreamMessage.

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

CreateTemporaryQueue Method

Create a temporary queue.

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

CreateTextMessage Method

Create a TextMessage.

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

Name	Description
Text	The string used to initialize this message.

Recover Method

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

```
QueueSession.Recover()
```

Rollback Method

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

```
QueueSession.Rollback()
```

Run Method

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

```
QueueSession.Run()
```

4.20.2 QueueSession Object Properties

The QueueSession object includes the following properties:

- **MessageListener Property** on page 79
- **Transacted Property** on page 79

MessageListener Property

This property is not currently supported.

Transacted Property

Returns an indication that the session is in transacted mode.

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

4.21 Session Object

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

4.21.1 Session Object Methods

The Session object includes the following methods:

- **Commit Method** on page 80
- **CreateBytesMessage Method** on page 80
- **CreateMapMessage Method** on page 80
- **CreateMessage Method** on page 80
- **CreateStreamMessage Method** on page 80
- **CreateTextMessage Method** on page 80
- **Recover Method** on page 80
- **Rollback Method** on page 81
- **Run Method** on page 81

Commit Method

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

```
Session.Commit
```

CreateBytesMessage Method

The CreateBytesMessage method creates a BytesMessage.

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

CreateMapMessage Method

The CreateMapMessage method creates a MapMessage.

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

CreateMessage Method

Create a Message.

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

CreateStreamMessage Method

Create a StreamMessage.

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

CreateTextMessage Method

Create a TextMessage.

Session.CreateTextMessage([*Text*])

Name	Description
Text	The string used to initialize this message.

Recover Method

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

```
Session.Recover
```

Rollback Method

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

```
Session.Rollback
```

Run Method

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

```
Session.Run
```

4.21.2 Session Object Properties

The Session object includes the following properties:

- **MessageListener Property** on page 81
- **Transacted Property** on page 81

MessageListener Property

This property is currently not supported.

Transacted Property

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

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

4.22 StreamMessage Object

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

4.22.1 StreamMessage Object Methods

The StreamMessage object includes the following methods:

- **Acknowledge Method** on page 82
- **ClearBody Method** on page 82
- **ClearProperties Method** on page 82
- **GetProperty Method** on page 82
- **ReadString Method** on page 84
- **Reset Method** on page 84
- **SetProperty Method** on page 84
- **WriteBoolean Method** on page 85

Acknowledge Method

Acknowledges the receipt of current and previous messages.

```
StreamMessage.acknowledge
```

ClearBody Method

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

```
StreamMessage.ClearBody
```

ClearProperties Method

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

```
StreamMessage.ClearProperties
```

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.

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.

ReadBoolean Method

Reads a Boolean value from the bytes message stream.

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

ReadByte Method

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

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

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.

ReadChar Method

Reads a Unicode character value from the bytes message stream.

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

ReadDouble Method

Reads a double from the bytes message stream.

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

ReadFloat Method

Reads a float from the bytes message stream.

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

ReadInt Method

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

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

ReadLong Method

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

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

ReadObject Method

Currently not supported.

ReadShort Method

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

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

ReadString Method

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

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

Reset Method

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

```
StreamMessage.Reset
```

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.

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.

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.

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.

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.

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.

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.

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.

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.

WriteObject Method

Currently not supported

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.

WriteString Method

Write a string to the message stream.

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

Name	Description
value	The String value that is written.

4.22.2 StreamMessage Object Properties

The StreamMessage object includes the following properties:

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
```

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
```

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.

Destination Property

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

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

Expiration Property

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

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

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
```

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.

Redelivered Property

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

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

ReplyTo Property

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

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

Timestamp Property

The TimeStamp property sets or returns the message timestamp.

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

Type Property

The Type property sets or returns the message type.

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

4.23 TemporaryQueue Object

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

4.23.1 TemporaryQueue Object Methods

The TemporaryQueue object includes the following methods:

- **Delete Method** on page 89
- **ToString Method** on page 90

Delete Method

The Delete method deletes the temporary queue.

```
TemporaryQueue.Delete
```

ToString Method

The ToString method returns a printed version of the queue name

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

4.23.2 TemporaryQueue Object Properties

The TemporaryQueue object includes the following property:

- **QueueName Property** on page 90

QueueName Property

The QueueName property returns the name of this queue.

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

4.24 TemporaryTopic Object

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

4.24.1 TemporaryTopic Object Methods

The TemporaryTopic object includes the following methods:

- **Delete Method** on page 90
- **ToString Method** on page 90

Delete Method

The Delete method deletes the temporary topic.

```
TemporaryTopic.Delete
```

ToString Method

The ToString method returns a printed version of the topic name

```
TemporaryTopic.ToString
```

4.24.2 TemporaryTopic Object Properties

The TemporaryTopic object includes the following property:

- **TopicName Property** on page 91

TopicName Property

The TopicName property returns the name of this topic.

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

4.25 TextMessage Object

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

4.25.1 TextMessage Object Methods

The TextMessage object includes the following methods:

- **Acknowledge Method** on page 91
- **ClearBody Method** on page 91
- **ClearProperties Method** on page 91
- **GetProperty Method** on page 92
- **PropertyExists Method** on page 92
- **SetProperty Method** on page 92

Acknowledge Method

Acknowledges the receipt of current and previous messages.

```
TextMessage.acknowledge
```

ClearBody Method

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

```
TextMessage.ClearBody
```

ClearProperties Method

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

```
TextMessage.ClearProperties
```

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.

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.

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.

4.25.2 TextMessage Object Properties

The TextMessage object includes the following methods:

- **CorrelationID Property** on page 93
- **CorrelationIDAsBytes Property** on page 93
- **DeliveryMode Property** on page 93
- **Destination Property** on page 93
- **Expiration Property** on page 93
- **MessageID Property** on page 94
- **Priority Property** on page 94
- **Redelivered Property** on page 94
- **ReplyTo Property** on page 94
- **Text Property** on page 94
- **Timestamp Property** on page 94
- **Type Property** on page 95

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
```

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
```

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.

Destination Property

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

```
TextMessage.Destination = Destination
Destination = TextMessage.Destination
```

Expiration Property

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

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

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
```

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.

Redelivered Property

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

```
TextMessage.Redelivered = Boolean
Boolean = TextMessage.Redelivered
```

ReplyTo Property

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

```
TextMessage.ReplyTo = Destination
Destination = TextMessage.ReplyTo
```

Text Property

The Text property sets or returns the string containing the message's data.

```
TextMessage.Text = String
String = TextMessage.Text
```

Timestamp Property

The TimeStamp property sets or returns the message timestamp.

```
TextMessage.Timestamp = Currency
Currency = TextMessage.Timestamp
```

Type Property

The Type property sets or returns the message type.

```
TextMessage.Type = String
String = TextMessage.Type
```

4.26 Topic Object

A Topic object encapsulates a Message Service specific topic name.

4.26.1 Topic Object Methods

The Topic object includes the following method:

- **ToString Method** on page 95

ToString Method

The ToString method returns a printed version of the topic name

```
Topic.ToString() As String
```

4.26.2 Topic Object Properties

The Topic object includes the following property:

- **TopicName Property** on page 95

TopicName Property

The TopicName property returns the name of this topic.

```
Topic.TopicName = String
String = Topic.TopicName
```

4.27 TopicConnection Object

A TopicConnection is an active connection to a Pub/Sub Message Service.

4.27.1 TopicConnection Object Methods

The TopicConnection object includes the following methods:

- **CreateTopicSession Method** on page 96
- **Start Method** on page 96
- **Stop Method** on page 96

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.

Start Method

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

```
TopicConnection.Start
```

Stop Method

The Stop method temporarily stops a Connection's delivery of incoming messages.

```
TopicConnection.Stop
```

4.27.2 TopicConnection Properties

The TopicConnection object includes the following properties:

- **ClientID Property** on page 97

- **MetaData Property** on page 97

ClientID Property

The ClientID property sets or returns a client identifier for this connection.

```
TopicConnection.ClientID = String
String = TopicConnection.ClientID
```

MetaData Property

This property is currently not supported.

4.28 TopicConnectionFactory Object

A client uses a TopicconnectionFactory to create TopicConnection with a Pub/Sub Message Service.

4.28.1 TopicConnectionFactory Object Methods

The TopicConnectionFactory object includes the following method:

- **CreateTopicConnection Method** on page 97

CreateTopicConnection Method

Create a topic connection with default user identity.

```
TopicConnectionFactory.createTopicConnection() As TopicConnection

TopicConnectionFactory.createTopicConnectioncreateTopicConnection(Str
ing user, String password) As TopicConnection
```

4.28.2 TopicConnectionFactory Properties

The TopicConnectionFactory object includes the following properties:

- **HostName Property** on page 98
- **Port Property** on page 98
- **PortOffset Property** on page 98

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
```

Port Property

The Port property sets or returns the port number that the JMS IQ Server is listening on, default value is 18007.

```
TopicConnectionFactory = Long
Long = TopicConnectionFactory
```

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
```

4.29 TopicPublisher Object

A Client uses a TopicPublisher for publishing messages on a topic.

4.29.1 TopicPublisher Object Methods

The TopicPublisher object includes the following method:

- **Publish Method** on page 98

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.

Name	Description
timeToLive	The message's lifetime (in milliseconds).
topic	The topic to publish this message to.

4.29.2 TopicPublisher Properties

The TopicPublisher object includes the following properties:

- **DeliveryMode Property** on page 99
- **DisableMessageID Property** on page 99
- **DisableMessageTimestamp Property** on page 99
- **Priority Property** on page 100
- **TimeToLive Property** on page 100
- **Topic Property** on page 100

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.

DisableMessageID Property

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

```
TopicPublisher.DisableMessageID = Boolean
Boolean = TopicPublisher.DisableMessageID
```

DisableMessageTimestamp Property

The DisableMessageTimestamp sets or returns an indication of whether message timestamps are disabled.

```
TopicPublisher.DisableMessageTimestamp = Boolean
Boolean = TopicPublisher.DisableMessageTimestamp
```

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.

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

Topic Property

The Topic property returns the topic associated with this publisher.

```
TopicPublisher.Topic = read-only
read-only = TopicPublisher.Topic
```

4.30 TopicRequestor Object

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

4.30.1 TopicRequestor Object Methods

The TopicRequestor object includes the following methods:

- **Create Method** on page 101
- **Request Method** on page 101

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.

Request Method

Send a request and wait for a reply

`TopicRequestor.Request(`*`message As message`*`) As message`

Name	Description
message	The message text.

4.31 TopicSession Object

A TopicSession provides methods for creating TopicPublishers, TopicSubscribers, and TemporaryTopics.

4.31.1 TopicSession Object Methods

The TopicSession object includes the following methods:

Commit Method

The Commit method commits all messages done in this transaction and releases any resources, currently held.

```
TopicSession.Commit
```

CreateBytesMessage Method

The CreateBytesMessage method creates a BytesMessage.

```
TopicSession.CreateBytesMessage() As BytesMessage
```

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.

CreateMapMessage Method

The CreateMapMessage method creates a MapMessage.

```
TopicSession.CreateMapMessage() As MapMessage
```

CreateMessage Method

Create a Message.

```
TopicSession.CreateMessage() As message
```

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.

CreateStreamMessage Method

Create a StreamMessage.

```
TopicSession.CreateStreamMessage() As StreamMessage
```

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.

CreateTemporaryTopic Method

The CreateTemporaryTopic method creates a temporary topic.

```
TopicSession.CreateTemporaryTopic() As TemporaryTopic
```

CreateTextMessage Method

The CreateTextMessage method creates TextMessage.

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

Name	Description
text	The string used to initialize this message.

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.

Recover Method

The Recover method creates a topic identity given a Topic name.

```
TopicSession.Recover
```

Rollback Method

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

```
TopicSession.Rollback
```

Run Method

The Run method is an optional method.

```
TopicSession.Run
```

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.

4.31.2 TopicSession Object Properties

The TopicSession object includes the following properties:

- **MessageListener Property** on page 104
- **Transacted Property** on page 105

MessageListener Property

This property is currently not supported.

Transacted Property

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

```
TopicSession.Transacted = Boolean
Boolean = TopicSession.Transacted
```

4.32 TopicSubscriber Object

A client uses a TopicSubscriber for receiving messages that have been published to a topic.

4.32.1 TopicSubscriber Object Methods

The TopicSubscriber object includes the following methods:

- **Close Method** on page 105
- **Receive Method** on page 105
- **ReceiveNoWait Method** on page 105

Close Method

Since a Message Service may allocate resources on behalf of a MessageConsumer, clients should close any unneeded resources.

```
TopicSubscriber.Close
```

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).

ReceiveNoWait Method

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

```
TopicSubscriber.ReceiveNoWait() As message
```

4.32.2 TopicSubscriber Object Properties

The TopicSubscriber object includes the following properties:

- **MessageListener Property** on page 106
- **MessageSelector Property** on page 106
- **NoLocal Property** on page 106
- **Topic Property** on page 106

MessageListener Property

This property is currently not supported.

MessageSelector Property

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

```
TopicSubscriber.MessageSelector = String
String = TopicSubscriber.MessageSelector
```

NoLocal Property

The NoLocal property returns the NoLocal attribute for this TopicSubscriber.

```
TopicSubscriber.NoLocal = Boolean
Boolean = TopicSubscriber.NoLocal
```

Topic Property

The Topic property returns the topic associated with this subscriber.

```
TopicSubscriber.Topic = Topic (read-only)
Topic (read-only) = TopicSubscriber.Topic
```

4.33 XAQueueConnection Object

An XAQueueConnection provides the same create options as QueueConnection. The only difference is that an XAQueueConnection is by definition transacted.

4.33.1 XAQueueConnection Object Methods

The XAQueueConnection object includes the following methods:

- **CreateQueueSession Method** on page 107

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.

CreateXAQueueSession Method

Create an XAQueueSession.

```
XAQueueConnection.CreateXAQueueSession() As XAQueueSession
```

Start Method

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

```
XAQueueConnection.Start
```

Stop Method

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

```
XAQueueConnection.Stop
```

4.33.2 XAQueueConnection Object Properties

The XAQueueConnection object includes the following properties:

- **ClientID Property** on page 108
- **MetaData Property** on page 108

ClientID Property

Returns or sets client identifier for this connection.

```
XAQueueConnection.ClientID = String
String = XAQueueConnection.ClientID
```

MetaData Property

Not currently supported.

4.34 XAQueueConnectionFactory Object

An XAQueueConnectionFactory provides the same create options as a QueueConnectionFactory, by definition, it is transacted.

4.34.1 XAQueueConnectionFactory Object Methods

The XAQueueConnectionFactory object includes the following methods:

- **CreateQueueConnection Method** on page 108
- **CreateXAQueueConnection Method** on page 108

CreateQueueConnection Method

Create a queue connection with a default user identity.

```
XAQueueConnectionFactory.CreateQueueConnection() As QueueConnection

XAQueueConnectionFactory.CreateQueueConnection(String user, String
password) As QueueConnection
```

CreateXAQueueConnection Method

Create an XA queue connection with a default user identity.

```
XAQueueConnectionFactory.CreateXAQueueConnection() As
XAQueueConnection
```

```
XAQueueConnectionFactory.CreateXAQueueConnection(String user, String
password) As XAQueueConnection
```

4.34.2 XAQueueConnectionFactory Object Properties

The XAQueueConnectionFactory object includes the following properties:

- **HostName Property** on page 109
- **Port Property** on page 109
- **PortOffset Property** on page 109

HostName Property

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

```
XAQueueConnectionFactory.HostName = String
String = XAQueueConnectionFactory.HostName
```

Port Property

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

```
XAQueueConnectionFactory.Port = Long
Long = XAQueueConnectionFactory
```

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
```

4.35 XAQueueSession Object

An XAQueueSession provides a regular QueueSession, which can be used to create QueueReceivers, QueueSenders, and QueueBrowsers.

4.35.1 XAQueueSession Object Methods

The XAQueueSession object includes the following methods:

- **Commit Method** on page 110
- **CreateBytesMessage Method** on page 110
- **CreateMapMessage Method** on page 110
- **CreateTextMessage Method** on page 110
- **Recover Method** on page 110
- **Rollback Method** on page 111

Commit Method

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

```
XAQueueSession.Commit
```

CreateBytesMessage Method

Create a BytesMessage.

```
XAQueueSession.CreateBytesMessage() As BytesMessage
```

CreateMapMessage Method

Create a MapMessage.

```
XAQueueSession.CreateMapMessage() As MapMessage
```

CreateMessage Method

Create a message.

```
XAQueueSession.CreateMessage() As message
```

CreateStreamMessage Method

Create a StreamMessage.

```
XAQueueSession.StreamMessage() As StreamMessage
```

CreateTextMessage Method

Create a TextMessage.

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

Name	Description
Text	The string used to initialize this message.

Recover Method

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

```
XAQueueSession.Recover()
```

Rollback Method

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

```
XAQueueSession.Rollback()
```

Run Method

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

```
XAQueueSession.Run()
```

4.35.2 XAQueueSession Object Properties

The XAQueueSession object includes the following properties:

- **MessageListener Property** on page 111
- **QueueSession Property** on page 111
- **Transacted Property** on page 111

MessageListener Property

This property is not currently supported.

QueueSession Property

Returns the queue session associated with this XAQueueSession.

```
XAQueueSession.QueueSession = QueueSession (read-only)
QueueSession (read-only)= XAQueueSession.QueueSession
```

Transacted Property

Returns an indication that the session is in transacted mode.

```
XAQueueSession.Transacted = Boolean
Boolean = XAQueueSession.Transacted
```

4.36 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).

4.36.1 XASession Object Methods

The XASession object includes the following methods:

Commit Method

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

```
XASession.Commit
```

CreateBytesMessage Method

The CreateBytesMessage method creates a BytesMessage.

```
XASession.CreateBytesMessage() As BytesMessage
```

CreateMapMessage Method

The CreateMapMessage method creates a MapMessage.

```
XASession.CreateMapMessage() As MapMessage
```

CreateMessage Method

Create a Message.

```
XASession.CreateMessage() As message
```

CreateStreamMessage Method

Create a StreamMessage.

```
XASession.CreateStreamMessage() As StreamMessage
```

CreateTextMessage Method

Create a TextMessage.

XASession.CreateTextMessage([*Text*])

Name	Description
Text	The string used to initialize this message.

Recover Method

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

```
XASession.Recover
```

Rollback Method

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

```
XASession.Rollback
```

Run Method

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

```
XASession.Run
```

4.36.2 XASession Object Properties

The XASession object includes the following properties:

- **MessageListener Property** on page 113
- **Transacted Property** on page 113

MessageListener Property

This property is currently not supported.

Transacted Property

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

```
XASession.Transacted = Boolean
Boolean = XASession.Transacted
```

4.37 XATopicConnection Object

An XATopicConnection provides the same create options as TopicConnection, but by definition is transacted.

4.37.1 XATopicConnection Object Methods

The XATopicConnection object includes the following methods:

- **CreateTopicSession Method** on page 114
- **Start Method** on page 114
- **Stop Method** on page 114

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.

Start Method

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

```
XATopicConnection.Start
```

Stop Method

The Stop method temporarily stops a Connection's delivery of incoming messages.

```
XATopicConnection.Stop
```

4.37.2 XATopicConnection Object Properties

The XATopicConnection object includes the following properties:

- **ClientID Property** on page 115
- **MetaData Property** on page 115

ClientID Property

The ClientID property sets or returns a client identifier for this connection.

```
XATopicConnection.ClientID = String
String = XATopicConnection.ClientID
```

MetaData Property

This property is currently not supported.

4.38 XATopicConnectionFactory Object

An XATopicConnectionFactory provides the same create options as TopicConnectionFactory, but by definition is transacted.

4.38.1 XATopicConnectionFactory Object Methods

The XATopicConnectionFactory object includes the following method:

- **CreateXATopicConnection Method** on page 115

CreateXATopicConnection Method

Create an XA topic connection with default user identity.

```
XATopicConnectionFactory.CreateTopicConnection() As TopicConnection

XATopicConnectionFactory.CreateTopicConnection(String user, String
password) As TopicConnection
```

4.38.2 XATopicConnectionFactory Object Properties

The XATopicConnectionFactory object includes the following properties:

- **HostName Property** on page 116
- **Port Property** on page 116
- **PortOffset Property** on page 116

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
```

Port Property

The Port property sets or returns the port number that the JMS IQ Server is listening on, default value is 18007.

```
XATopicConnectionFactory = Long
Long = XATopicConnectionFactory
```

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
```

4.39 XATopicSession Object

An XA TopicSession provides a regular TopicSession which can be used to create TopicSubscribers and TopicPublishers.

4.39.1 XATopicSession Object Methods

The XATopicSession object includes the following methods:

Commit Method

The Commit method commits all messages done in this transaction and releases any resources, currently held.

```
XATopicSession.Commit
```

CreateBytesMessage Method

The CreateBytesMessage method creates a BytesMessage.

```
XATopicSession.CreateBytesMessage() As BytesMessage
```

CreateMapMessage Method

The CreateMapMessage method creates a MapMessage.

```
XATopicSession.CreateMapMessage() As MapMessage
```

CreateMessage Method

Create a Message.

```
XATopicSession.CreateMessage() As message
```

CreateStreamMessage Method

Create a StreamMessage.

```
XATopicSession.CreateStreamMessage() As StreamMessage
```

CreateTextMessage Method

The CreateTextMessage method creates TextMessage.

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

Name	Description
text	The string used to initialize this message.

Recover Method

The Recover method creates a topic identity given a Topic name.

```
XATopicSession.Recover
```

Rollback Method

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

```
TopicSession.Rollback
```

Run Method

The Run method is an optional method

```
TopicSession.Run
```

4.39.2 XATopicSession Object Properties

The XATopicSession object includes the following properties:

- **MessageListener Property** on page 118
- **TopicSession Property** on page 118
- **Transacted Property** on page 118

MessageListener Property

This property is currently not supported.

TopicSession Property

Returns the topic session associated with this XATopicSession.

```
XATopicSession.TopicSession = TopicSession (read-only)
TopicSession (read-only) = TopicSession.TopicSession
```

Transacted Property

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

```
TopicSession.Transacted = Boolean
Boolean = TopicSession.Transacted
```

4.40 COM+ Error Codes

Common error codes in COM+ APIs are given below.

4.40.1 IErrorInfo Methods

Table 7 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.

4.40.2 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 8 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.

4.40.3 Error Value Constants

Error Value Constant	Explanation
Const msErrGeneral = 768 (&H300)	JMS exception, unspecified.
Const msErrReAlloc = 769 (&H301)	A JMS exception occurred as a result of memory reallocation.
Const msErrMalloc = 770 (&H302)	A JMS exception occurred as a result of memory allocation.
Const msErrConnection = 771 (&H303)	A JMS exception occurred in setting up a connection.
Const msErrCreation = 772 (&H304)	A JMS exception occurred while creating a JMS object.
Const msErrCloseSocket = 773 (&H305)	A JMS exception occurred because of a closed socket.
Const msErrMessageEOF = 774 (&H306)	Processing ended because the BytesMessage or StreamMessage ended unexpectedly.
Const msErrMessageNotReadable = 775 (&H307)	Processing ended because the message could not be read.
Const msErrMessageNotWriteable = 776 (&H308)	Processing ended because the message could not be written.

Error Value Constant	Explanation
Const msErrMessageFormat = 777 (&H309)	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.
Const msErrTransactionRolledBack = 778 (&H30A)	The attempt to commit the session was unsuccessful of a transaction being rolled back.
Const msErrIllegalState = 779 (&H30B)	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.
Const msErrInvalidDestination = 780 (&H30C)	Processing ended because the destination could not be understood or was found to be invalid.
Const msErrNotImplemented = 781 (&H30D)	Processing ended because a feature or interface was not implemented.
Const msErrIndexOutOfBounds = 782 (&H30E)	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.
Const msErrNullPointer = 783 (&H30F)	Processing ended because the pointer in a case where an object was required.
Const msErrInvalidClientID = 784 (&H310)	Processing ended because the connection's client ID was rejected by the provider.
Const msErrInvalidSelector = 785 (&H311)	Processing ended because the message selector was found to be syntactically invalid.
Const msErrSecurity = 786 (&H312)	Processing was ended by JMS Security — for example, the provider rejected a name/password combination submitted by a client.
Const msErrResourceAllocation = 787 (&H313)	Processing ended because of the provider was unable to allocate resources required for the method/function.
Const msErrTransactionInProgress = 788 (&H314)	Processing ended because a transaction was in progress.

Chapter 5

Working with the COM+ API Samples

The eGate API Kit for JMS IQ Manager includes code and Project samples. This chapter describes how to use the code samples to build a sample COM+ application for JMS IQ Manager, and then describes how to run the sample application through the JMS Server.

What's in This Chapter

5.1 About the COM+ Samples

The eGate API Kit provides COM+ code samples and Enterprise Designer Project samples designed to work together to demonstrate different types of JMS messaging using a COM+ client and eGate Integrator. The sample Projects provide examples of the following messaging types:

- Publish/subscribe (queues or topics)
- Request-reply (queues or topics)
- Message selector (topics)
- Publish/subscribe using XA (topics)

The sample file, **eGateAPIKit_Sample.zip**, contains the **.zip** files listed in Table 9. The table describes what each **.zip** file contains.

Table 9 eGate API Kit Samples

File Name	Contents
CodeSamples.zip	The sample code files for use on Windows.
CodeSamplesUNIX.tar	The sample code files for use on UNIX operating systems.

Table 9 eGate API Kit Samples

File Name	Contents
Sample_Project.zip	A Java CAPS Project that you can import into Enterprise Designer.

5.2 Implementing the Java CAPS Projects

The sample Java CAPS Projects include one Project and several sub-Projects, each used to demonstrate a different type of JMS messaging. Each Project uses one of three available pass-through Collaborations to deliver messages between senders and receivers or between publishers and subscribers.

Before continuing, make sure you have downloaded the sample file as described in **"Installing the eGate API Kit" on page 18**. Implementing the sample Projects consists of the following steps.

5.2.1 Importing the Sample Project

To work with the sample Projects for Enterprise Designer, you first need to import the Projects into Enterprise Designer.

To import the sample Project into Enterprise Designer

1. If you have not already done so, extract **eGateAPIKit_Sample.zip**.
2. Start Enterprise Designer.
3. From the Repository context menu, select **Import Project**.
4. A message box appears, prompting you to save any unsaved changes to the Repository.
   - A If you want to save your changes and have not already done so, click **No**. Save your changes, and then re-select **Import Project**.
   - B If you have saved all changes, click **Yes**.
5. Click the **Browse** button to display the Open File dialog.
6. Locate and select **Sample_Project.zip**, located in the directory in which you extracted **eGateAPIKit_Sample.zip**.
7. Click **Open** to select the file.

   The Import Manager dialog appears.
8. Click **Import** to import the file.

Note: *An error message might appear, stating that certain APIs are missing. This error is not serious. Click* ***Continue*** *to proceed with the import.*

The Import Status message box appears after the file is imported successfully.

9 Click **OK** to close the message box.

10 When you are finished importing files, click **Close** to close the Import Manager dialog. The Project Explorer is automatically refreshed from the Repository.

5.2.2 Creating the Environment

In order to deploy the Projects to a Logical Host, you must create an Environment used by all sub-Projects.Use the Environment Explorer of Enterprise Designer to create a new Environment and Logical Host. The Logical Host must include a JMS server and application server. For more information about Environments, see the *Sun SeeBeyond eGate Integrator User's Guide* or the *Sun SeeBeyond eGate Integrator System Administration Guide*.

5.2.3 Deploying the Projects

For each sample sub-Project, you must create a Deployment Profile, and then build and deploy the Project. You can use the Automap feature of the Deployment Profile to map each Project component to its corresponding Environment component.

Before deploying the sub-Projects, make sure the Logical Host for the sample applications is started. For more information about Deployment Profiles, see the *Sun SeeBeyond eGate Integrator User's Guide* or the *Sun SeeBeyond eGate Integrator System Administration Guide*.

5.3 Building the Sample COM+ Application

The sample COM+ files are provided in both Microsoft Visual Studio 6.0 format and Microsoft .NET 2003 format. Before building the sample applications, you need to configure the environment. Follow these steps to build the sample applications:

- **Setting up the Directory Structure** on page 123
- **Configuring the Sample Environment** on page 123
- **Building the Sample Applications** on page 123

Table 10 lists each messaging type demonstrated in the samples along with their corresponding folder names and the name of the queue or topic you need to specify for each sample.

Table 10 COM+ Sample Information

Messaging Type Sample Directory Name	Executable	Sending Topic or Queue	Receiving Topic or Queue
Queue Send and Receive \Point2Point_Sample	p2p.exe	P2PSample	eGateP2PSample
Queue Requestor \QueueRequestor_Sample	queuerequestor.exe	QueueRequestor Sample	OutputQueue
Topic Publish and Subscribe \PublishSubscribe_Sample	pubsub.exe	PubSubSample	eGatePubSubSample
Topic Requestor \TopicRequestor_Sample	topicrequest.exe	TopicRequestor Sample	OutputTopic
Topic Selector \MessageSelector_Sample	messageselector.exe	Selector	eGateSelector
XA Publish and Subscribe \XA_Sample	xatest	XAPubSubSample	eGateXAPubSubSample
CRM \CRM_Sample	CRMClient.exe	XAPubSubSample	eGateXAPubSubSample

5.3.1 Setting up the Directory Structure

The sample files must be located in a specific directory in relation to the API kit library files.

To set up the directory structure

1. Navigate to the location where you extracted **eGateAPIKIT_Sample.zip**.
2. From the extracted files, extract **CodeSamples.zip**.
3. In the extracted folders, navigate to the \apikit folder and copy the \com folder to the location where you installed the eGate API Kit at the same level as the \jms folder. The \com folder contains all of the sample files.

5.3.2 Configuring the Sample Environment

In order to compile the COM+ sample client applications (or any client applications you create), you must edit the PATH variable by adding the path to the library files you downloaded during installation (see **"Post-Installation Instructions" on page 19**). Make sure this has been completed before performing the following steps.

5.3.3 Building the Sample Applications

All samples provided with the toolkit have make files, and the project files can be edited and compiled in both Visual Studio 6.0 and .NET 2003 except XA_Sample and CRM_Sample, which require Visual Studio 6.0.

The CRM sample requires additional setup. To build the CRM application, follow the steps outlined in **"Building the CRM Sample Application" on page 124**.

To build the sample COM+ applications

Important: *Make sure to build the CRM sample before building the XA sample. The XA sample requires the newly-built* ***CRMTest.dll*** *file to compile.*

1. To access the sample project you want to compile, do one of the following:
   - To open the project in Visual Studio, open the **.vbp** file for the sample (see **Table 10 on page 123** for sample file locations).
   - To open the project in .NET, open the **.sln** file for the sample. This file is located in the <project_name>.NET folder in the sample directory (see **Table 10 on page 123** for sample file locations).
2. Before building a sample, change the hostname and port number values where ever they occur in the project.
3. Save your changes to the project, and then build the project.

5.4 Building the CRM Sample Application

There are three primary steps to setting up the CRM sample application. Perform the steps in the following order:

- **Creating a Database for the CRM Sample** on page 124
- **Configuring and Building the CRM Sample** on page 126
- **Creating the CRM Sample Application** on page 127

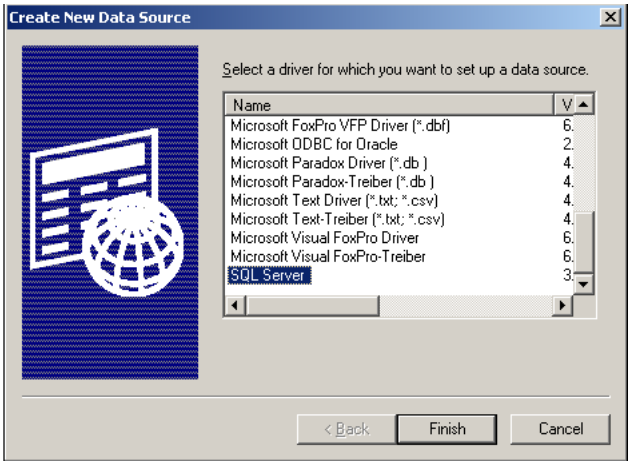
5.4.1 Creating a Database for the CRM Sample

In order to use the Compensating Resource Manager (CRM) samples provided, you must create a SQL Server database named "CRM".

Creating a 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 the Control Panel, select **Administrative Tools** and then double-click **Data Sources (ODBC)**.
5. On the ODBC Data Source Administrator window, click **Add**, and then select SQL_Server. Click **Finish** to continue.

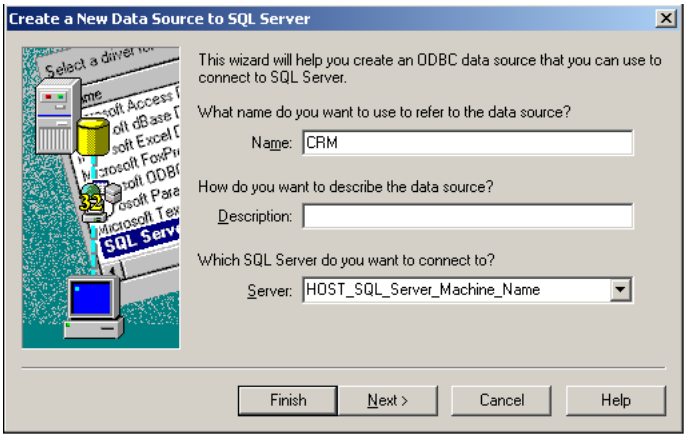
Figure 21 SQL Database Source



6 Provide the name of the data source, a description if desired, and the machine name on which SQL Server is running. Click **Next** to continue.

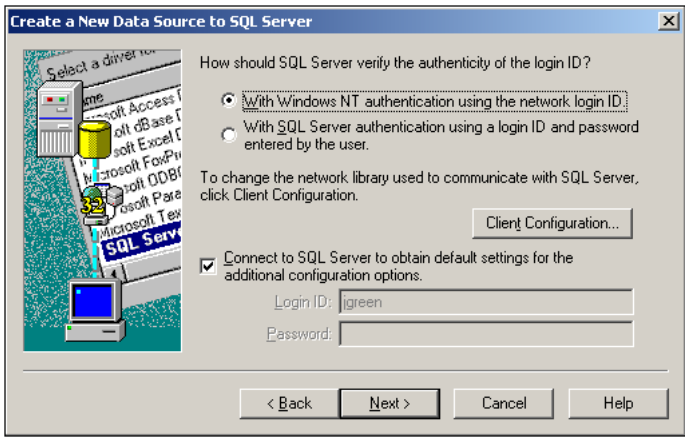
Important: *You will not be able to continue until a successful connection is made.*

Figure 22 SQL Data source



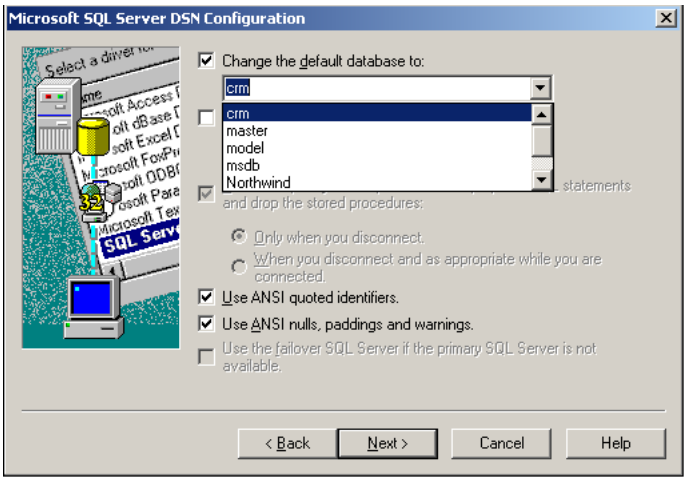
7 Select **With Windows authentication using the network logon ID**, select **Connect to SQL Server to obtain default settings for the additional configuration options**, and then click **Next**.

Figure 23 Login Settings



8 For the default database, select the database you created earlier from the drop-down list. Click **Next** to continue.

Figure 24 Default SQL Server Database



9 Click **Finish**.

5.4.2 Configuring and Building the CRM Sample

Two sample files are used to run the CRM sample. The samples can be found in the location where you extracted the sample files under **\CodeSamples\apikit\com**. The files used are:

- \CRM_Sample\CRMDLL\CRMTest.vbp
- \CRM_Sample\CRMDLL\CRMTest.dll

To configure the CRM

1 Using Visual Studio 6.0, open **CRMTest.vbp**.

2. Follow the comments in the code of the following files to modify the sample to your system requirements. Make sure to customize all instances of hostname and port.
   - InsertMessage.cls
   - TwoTasks.cls
   - TopicTask.cls
   - QueueTasks.cls
3. Save your changes and recompile the sample application.
4. Copy **CRMclient.exe** (located in the CRM_Sample folder) to the machine on which the external code is to run.
5. On the machine where you copied **CRMclient.exe**, register the file **CRMTest.dll** into the Windows registry by running the following command:

```
regsvr32 your_path_location\CRMTest.dll
```

5.4.3 Creating the CRM Sample Application

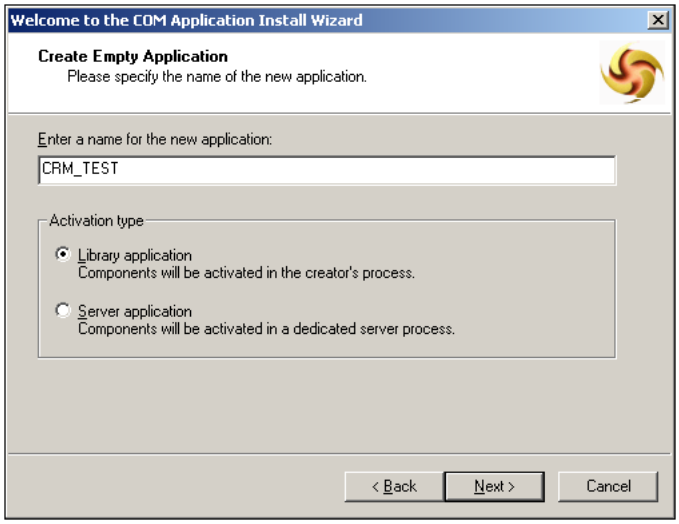
Once the CRM sample has been recompiled, use the Windows Component Services administrative tools to create the COM+ Application.

To create the CRM sample application

1. From the Control Panel, select **Administrative Tools** and then **Component Services**.
2. Expand the Component Services folder, and then right click on C**OM+ Applications**.

   Select **New** and then select **Application**. The COM+ Application Install Wizard appears.
3. On the Welcome page, click **Next**, and then select **Create an empty application**.
4. Enter **CRM_TEST** as the name of the new application (you can use any name), and then select **Library application** as the Activation Type.

Figure 25 CRM_TEST Application



5 Click **Next**, and then click **Finish**.

6 On the Component services window, expand the **CRM_TEST** component.

7 Right-click the **Components** folder, click **New**, and then click **Component**. The COM+ Component Install Wizard appears.

8 On the Welcome window, click **Next**, and then click **Install new component(s)**.

9 Browse to the location of the recently compiled **CRMTest.dll** and click **Open**.

10 Accept the remainder of the default settings, and then click **Next** and **Finish**.

5.5 Running the Sample COM+ Applications

There are several different sample applications you can run. Each sends and receives a simple message, using a Collaboration in the Java CAPS sample Project to transfer the message. You can use Enterprise Manager to monitor the activity of the Projects.

For information about the project names and locations referenced below, see **Table 10 on page 123**.

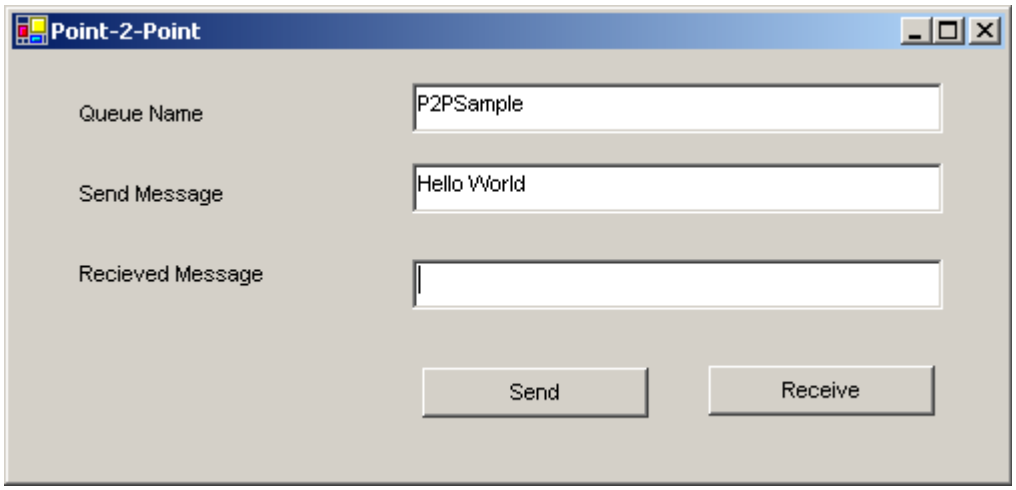
To run a send/receive or publish/subscribe sample application

1 Navigate to the directory containing the sample you want to run.

2 Double-click the executable file. If you compiled using .NET, the executable is located in the sample directory in \<project_name>.NET\bin. If you compiled using Visual Studio, the executable is in the sample directory.

A dialog appears.

Figure 26 Point 2 Point Sample Dialog



3 On the dialog, enter the name of the producer topic or queue, change the message if desired, and then click **Publish** or **Send**.

4 Change the queue or topic name to the consumer topic or queue, and then click **Receive**.

A message appears stating whether the message was processed.

To run the selector sample application

1 Navigate to the directory containing the sample you want to run.

2 Double-click the executable file. If you compiled using .NET, the executable is located in the sample directory in \<project_name>.NET\bin. If you compiled using Visual Studio, the executable is in the sample directory.

A dialog appears.

3 On the dialog, enter the name of the producer topic, change the message if desired, select the **Match** option, and then click **Publish**.

4 Change the topic name to the consumer topic, and then click **Receive**.

The text of the message appears in the **Received Message** field.

To run a requestor sample application

1 Navigate to the directory containing the sample you want to run.

2 Double-click the executable file. If you compiled using .NET, the executable is located in the sample directory in \<project_name>.NET\bin. If you compiled using Visual Studio, the executable is in the sample directory.

A dialog appears.

3 On the dialog, enter the name of the requester topic or queue, change the message if desired, and then click **Start**.

The text of the message appears in the **Received Message** field.

To run an XA sample application

Note: *The **CRMTest.dll** file must be registered on the machine on which the XA sample resides before you can run the XA sample.*

1. Navigate to the directory containing the sample you want to run.
2. Double-click the executable file (the executable is in the sample directory).

   A dialog appears.
3. On the dialog, do the following:
   - A Enter the name of the producer topic or queue.
   - B Change the message if desired.
   - C Select **Topic** to publish via a Topic; deselect **Topic** to send via a Queue.
   - D Select **Commit** to send the message.
   - E Click **Publish**.
4. Repeat steps 2 and 3 with the following changes:
   - Enter the name of the *consumer* topic or queue.
   - Click **Receive** instead of **Publish**.

To run the CRM sample application

Note: *Make sure you have completed all of the steps in* **"Building the CRM Sample Application" on page 124**.

1. Navigate to the directory containing the sample you want to run.
2. Double-click the executable file (he executable is in the sample directory).

   A dialog appears.
3. On the dialog, do the following:
   - A Enter the name of the producer topic or queue.
   - B Change the message if desired.
   - C Select **Commit** to send the message.
   - D Click **Publish**.
4. Repeat steps 2 and 3 with the following changes:
   - Enter the name of the *consumer* topic or queue.
   - Click **Receive** instead of **Publish**.

Index

Symbols

A

B

C

D

E

G

H

I

J

L

M

N

O

P

Q

R

S

T

U

V

W

X