SUN SEEBEYOND

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

Release 5.1.2



Copyright © 2006 Sun Microsystems, Inc., 4150 Network Circle, Santa Clara, California 95054, U.S.A. All rights reserved. Sun Microsystems, Inc. has intellectual property rights relating to technology embodied in the product that is described in this document. In particular, and without limitation, these intellectual property rights may include one or more of the U.S. patents listed at http://www.sun.com/patents and one or more additional patents or pending patent applications in the U.S. and in other countries. U.S. Government Rights - Commercial software. Government users are subject to the Sun Microsystems, Inc. standard license agreement and applicable provisions of the FAR and its supplements. Use is subject to license terms. This distribution may include materials developed by third parties. Sun. Sun Microsystems, the Sun logo, lava, Sun lava Composite Application Platform Suite, SeeBeyond, eGate, eInsight, eVision, eTL, eXchange, eView, eIndex, eBAM, eWay, and JMS are trademarks or registered trademarks of Sun Microsystems, Inc. in the U.S. and other countries. All SPARC trademarks are used under license and are trademarks or registered trademarks of SPARC International, Inc. in the U.S. and other countries. Products bearing SPARC trademarks are based upon architecture developed by Sun Microsystems, Inc. UNIX is a registered trademark in the U.S. and other countries, exclusively licensed through X/Open Company, Ltd. This product is covered and controlled by U.S. Export Control laws and may be subject to the export or import laws in other countries. Nuclear, missile. chemical biological weapons or nuclear maritime end uses or end users, whether direct or indirect, are strictly prohibited. Export or reexport to countries subject to U.S. embargo or to entities identified on U.S. export exclusion lists, including, but not limited to, the denied persons and specially designated nationals lists is strictly prohibited.

Copyright © 2006 Sun Microsystems, Inc., 4150 Network Circle, Santa Clara, California 95054, Etats-Unis. Tous droits réservés. Sun Microsystems, Inc. détient les droits de propriété intellectuels relatifs à la technologie incorporée dans le produit qui est décrit dans ce document. En particulier, et ce sans limitation, ces droits de propriété intellectuels peuvent inclure un ou plus des brevets américains listés à l'adresse http://www.sun.com/patents et un ou les brevets supplémentaires ou les applications de brevet en attente aux Etats - Unis et dans les autres pays. L'utilisation est soumise aux termes de la Licence. Cette distribution peut comprendre des composants développés par des tierces parties. Sun, Sun Microsystems, le logo Sun, Java, Sun Java Composite Application Platform Suite, Sun, SeeBeyond, eGate, eInsight, eVision, eTL, eXchange, eView, eIndex, eBAM et eWay sont des marques de fabrique ou des marques déposées de Sun Microsystems, Inc. aux Etats-Unis et dans d'autres pays. Toutes les marques SPARC sont utilisées sous licence et sont des marques de fabrique ou des marques déposées de SPARC International, Inc. aux Etats-Unis et dans d'autres pays. Les produits portant les marques SPARC sont basés sur une architecture développée par Sun Microsystems, Inc. UNIX est une marque déposée aux Etats-Unis et dans d'autres pays et licenciée exclusivement par X/Open Company, Ltd. Ce produit est couvert à la législation américaine en matière de contrôle des exportations et peut être soumis à la règlementation en vigueur dans d'autres pays dans le domaine des exportations et importations. Les utilisations, ou utilisateurs finaux, pour des armes nucléaires, des missiles, des armes biologiques et chimiques ou du nucléaire maritime, directement ou indirectement, sont strictement interdites. Les exportations ou réexportations vers les pays sous embargo américain, ou vers des entités figurant sur les listes d'exclusion d'exportation américaines, y compris, mais de manière non exhaustive, la liste de personnes qui font objet d'un ordre de ne pas participer, d'une façon directe ou indirecte, aux exportations des produits ou des services qui sont régis par la législation américaine en matière de contrôle des exportations et la liste de ressortissants spécifiquement désignés, sont rigoureusement interdites.

Part Number: 819-7434-10 Version 20061011155507

Chapter 1	
Introduction	12
About This Document What's in This Document Intended Audience Text Conventions Screenshots	12 12 12 13 13
Related Documents	13
Sun Microsystems, Inc. Web Site	13
Documentation Feedback	14
Chapter 2	
Installing the eGate API Kit	15
Supported Operating Systems	15
System Requirements	15
Supported Compilers	15
Installing the eGate API Kit	16
Post-Installation Instructions	17
Chapter 3	
JMS and COM+ Implementation Overview	18
About the Sun SeeBeyond JMS IQ Manager The Java CAPS JMS Interface Java CAPS Project Considerations Viewing JMS IQ Manager Port Numbers	18 18 20 20
About the Java Messaging Service JMS Messages Message Header Fields Message Properties Message Body (Payload) JMS Messaging Types	20 21 21 22 22 22

Publish/Subscribe Projects Point-to-Point Projects Paguest Poply Projects	22 23 24
Request-Reply Projects About the COM+ Interface	24 25
Creating Destinations	25
XA Compliance Message Selectors	25 25
Multi-Threaded Apartments	25
Sample Code	26
The Compensating Resource Manager (CRM)	26
About Compensating Resource Managers	26
CRM Architecture	26
Two-phase Commit Protocol	27
Implementing the Compensating Resource Manager	28
Configuring the Compensating Resource Manager	28
Step 1: Add a New Component Application	28
Step 2: Installing the stc_mscom Component	31
Step 3: Configuring the stc_mscom Component	33
Step 4: Configuring the STC_MSCOM.Compensator Step 5: Configuring the STC_MSCOM XA Connection Factories	33 35
Chapter 4	
COM+ Object Reference	37
About the Java CAPS JMS Interface for Applications in COM+	38
The COM+ Object Model	38
Wrapper Functions for COM+ Applications	39
COM+ API Correspondence with JMS	40
Viewing the COM+ API	41
Creating Destinations	42
COM+ Constants	42
AcknowledgeMode Constants	42
DeliveryMode Constants	43
Message Constants	43
Priority Constants	44
BytesMessage Object	44
BytesMessage Properties	45
BytesMessage Methods	45
Acknowledge	46
ClearBody	46
ClearProperties	47
GetProperty	47
PropertyExists PeadPeadean	47
ReadBoolean ReadByte	48 48
ReadBytes	49
ReadChar	49
ReadDouble	40

ReadFloat	50
ReadInt	50
ReadLong	50
ReadShort	51
ReadUnsignedByte	51
ReadUnsignedShort	51
ReadUTF	52
Reset	52
SetProperty	53
WriteBoolean	53
WriteByte	53
WriteBytes	54
WriteChar	54
WriteDouble	55
WriteFloat	55
WriteInt	55
WriteLong	56
WriteObject	56
WriteShort	56
WriteUTF	57
Connection Object	57
Connection Properties	57
Connection Methods	57
Start	58
Stop	58
ConnectionFactory Object	58
ConnectionFactory Properties	58
ConnectionMetaData Object	59
MapMessage Object	59
MapMessage Properties	59
MapMessage Object Methods	60
Acknowledge	61
ClearBody	61
ClearProperties	61
GetBoolean	62
GetByte	62
GetBytes	62
GetChar	63
GetDouble	63
GetFloat	64
GetInt	64
GetLong	65
GetObject	65
GetProperty	65
GetShort	66
GetString	66
ItemExists	66
PropertyExists	67
	07
SetBoolean	67

SetChar	69
SetDouble SetFloat	69 69
SetInt	70
SetLong	70
SetObject	71
SetProperty	71
SetShort	71
SetString	72
message Object	72
message Properties	72
message Methods Acknowledge	73 73
ClearBody	73
ClearProperties	74
GetProperty	74
PropertyExists	75
SetProperty Method	75
MessageConsumer Object	76
MessageConsumer Properties	76
MessageConsumer Methods	76
Close	76 76
ReceiveMessage ReceiveNoWait	76
MessageListener Object	77
MessageProducer Object MessageProducer Properties	77 78
•	78
Queue Object Queue Properties	78
Queue Methods	79
ToString	79
QueueBrowser Object	79
QueueConnection Object	79
QueueConnection Properties	79
QueueConnection Methods	80
CreateQueueSession	80
Start	80
Stop	81
QueueConnectionFactory Object	81
QueueConnectionFactory Properties	81
QueueConnectionFactory Methods CreateQueueConnection	81 82
CreateQueueConnectionEx	82
QueueReceiver Object	82
QueueReceiver Properties	83
QueueReceiver Methods	83
Close	83
Receive	83

ReceiveNoWait	84
QueueRequestor Object	84
QueueRequestor Methods	84
Close	85
Create	85
Request	85
QueueSender Object	86
QueueSender Properties	86
QueueSender Methods	87
Send	87
QueueSession Object	88
QueueSession Properties	88
QueueSession Object Methods	88
Commit	88
CreateBytesMessage	89
CreateMapMessage	89
CreateMessage	89
CreateQueue	90
CreateReceiver	90
CreateSender	91
CreateStreamMessage	91
CreateTemporaryQueue	91
CreateTextMessage	92
Recover	92
Rollback	92
Run	93
session Object	93
session Properties	93
session Methods	94
Commit	94
CreateBytesMessage	94
CreateMapMessage	95
CreateMessage	95
CreateStreamMessage	95
CreateTextMessage	96
Recover	96
Rollback	96
Run	97
StreamMessage Object	97
StreamMessage Properties	97
StreamMessage Methods	98
Acknowledge	99
ClearBody	99
ClearProperties	99
GetProperty Property Spirits	100
PropertyExists	100
ReadBoolean	101
ReadByte BoodBytes	101 101
ReadBytes ReadChar	101 102
ReadDouble Method	102
INCOULOUS MEUTON	107

ReadFloat	102
ReadInt	103
ReadLong Method	103
ReadObject	103
ReadShort	104
ReadString	104
Reset	104
SetProperty	105
WriteBoolean	105
WriteByte	105
WriteBytes	106
WriteChar	106
WriteDouble	107
WriteFloat	107
WriteInt	107
WriteLong	108
WriteObject	108
WriteShort	108
WriteString	109
TemporaryQueue Object	109
TemporaryQueue Properties	109
TemporaryQueue Object Methods	110
Delete	110
ToString	110
TemporaryTopic Object	110
TemporaryTopic Properties	111
TemporaryTopic Methods	111
Delete	111
ToString	111
<u> </u>	
TextMessage Object	112
TextMessage Properties	112
TextMessage Methods	113
Acknowledge	113
ClearBody	113
ClearProperties	113
GetProperty Property Sylicts	114 114
PropertyExists SetProperty	115
• ,	
Topic Object	115
Topic Properties	115
Topic Methods	115
ToString	116
TopicConnection Object	116
TopicConnection Properties	116
TopicConnection Methods	116
CreateTopicSession	116
Start	117
Stop	117
•	
TopicConnectionFactory Object	118
TopicConnectionFactory Properties	118

TopicConnectionFactory Methods CreateTopicConnection	118 118
CreateTopicConnectionEx	119
TopicPublisher Object	119
TopicPublisher Properties	119
TopicPublisher Methods	120
Publish	120
TopicRequestor Object	121
TopicRequestor Methods	121
Close	121
Create	121
Request	122
TopicSession Object	122
TopicSession Properties	123
TopicSession Methods	123
Commit	123
CreateBytesMessage CreateDurableSubscriber	124 124
CreateMapMessage	125
CreateMessage	125
CreatePublisher	125
CreateStreamMessage	126
CreateSubscriber	126
CreateTemporaryTopic	127
CreateTextMessage	127
CreateTopic Recover	127
Rollback	128 128
Run	129
Unsubscribe	129
TopicSubscriber Object	129
TopicSubscriber Properties	130
TopicSubscriber Methods	130
Close	130
Receive	131
ReceiveNoWait	131
XAQueueConnection Object	131
XAQueueConnection Properties	132
XAQueueConnection Methods	132
CreateQueueSession	132
CreateXAQueueSession	133
Start	133 133
Stop	
XAQueueConnectionFactory Object	134
XAQueueConnectionFactory Object Properties	134
XAQueueConnectionFactory Methods	134 134
CreateQueueConnection CreateQueueConnectionEx	134
CreateXAQueueConnection	135
CreateXAQueueConnectionEx	135

XAQueueSession Object	136
XAQueueSession Properties	136
XAQueueSession Methods	136
Commit	137
CreateBytesMessage	137
CreateMapMessage	137
CreateMessage	138
CreateStreamMessage	138
CreateTextMessage	138
Recover	139
Rollback	139
Run	140
XASession Object	140
XASession Object Properties	140
XASession Object Methods	140
Commit	141
CreateBytesMessage	141
CreateMapMessage	141
CreateMessage	142
CreateStreamMessage	142
CreateTextMessage	142
Recover	143
Rollback	143
Run	144
XATopicConnection Object	144
XATopicConnection Properties	144
XATopicConnection Methods	144
CreateTopicSession	145
CreateXATopicSession	145
Start	146
Stop	146
XATopicConnectionFactory Object	146
XATopicConnectionFactory Properties	147
XATopicConnectionFactory Methods	147
CreateTopicConnection	147
CreateTopicConnectionEx	148
CreateXATopicConnection	148
CreateXATopicConnectionEx	148
·	149
XATopicSession Object	
XATopicSession Properties	149 149
XATopicSession Methods Commit	150
CreateBytesMessage	150
,	150
CreateMapMessage CreateMessage	150
CreateStreamMessage	151
CreateTextMessage	151
Recover	151
Rollback	152
Run	153
Error Codes	153

IErrorInfo Methods HRESULT Errors Error Value Constants	153 153 154
Chapter 5	
Working with the COM+ API Samples	156
About the COM+ Samples	156
Implementing the Java CAPS Projects Importing the Sample Project Creating the Environment Deploying the Projects	157 157 158 158
Building the Sample COM+ Application Setting up the Directory Structure Configuring the Sample Environment Building the Sample Applications	158 159 159 159
Building the CRM Sample Application Creating a Database for the CRM Sample Configuring and Building the CRM Sample Creating the CRM Sample Application	160 160 162 163
Running the Sample COM+ Applications	164
Index	167

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

- About This Document on page 12
- Related Documents on page 13
- Sun Microsystems, Inc. Web Site on page 13
- Documentation Feedback on page 14

1.1 About This Document

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

1.1.1 What's in This Document

This document includes the following chapters:

- Chapter 2 "Installing the eGate API Kit" on page 15 describes how to install the eGate API Kit and its samples.
- Chapter 3 "JMS and COM+ Implementation Overview" on page 18 gives information about the JMS IQ Manager and how the COM+ API interfaces with it.
- Chapter 4 "COM+ Object Reference" on page 37 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 156 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 13
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:

 $CAPS_docs feedback@sun.com$

Installing the eGate API Kit

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

What's in This Chapter

- Supported Operating Systems on page 15
- System Requirements on page 15
- Supported Compilers on page 15
- Installing the eGate API Kit on page 16
- Post-Installation Instructions on page 17

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 eGate API Kit COM+ API is compatible with .Net 2003 version 7.1 on a Windows platform. If you use a different compiler, be aware that some compilers are incompatible with eGate.

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

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

- About the Sun SeeBeyond JMS IQ Manager on page 18
- About the Java Messaging Service on page 20
- About the COM+ Interface on page 25
- The Compensating Resource Manager (CRM) on page 26

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.

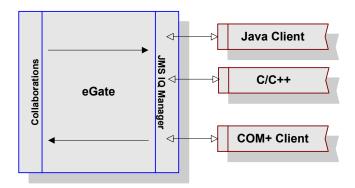
3.1.1 The Java CAPS JMS Interface

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

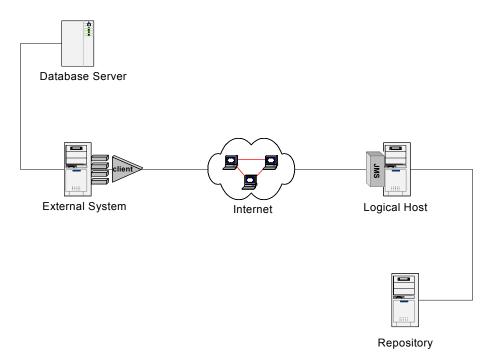
Figure 1 illustrates the communication between each component.

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.

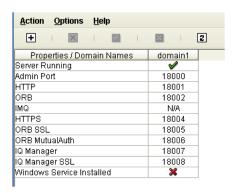
3.1.3 Viewing JMS IQ Manager Port Numbers

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 applicationspecific 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.

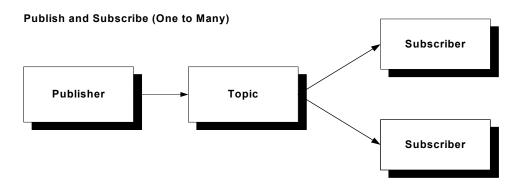
- Publish/Subscribe Projects on page 22
- Point-to-Point Projects on page 23
- Request-Reply Projects on page 24

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

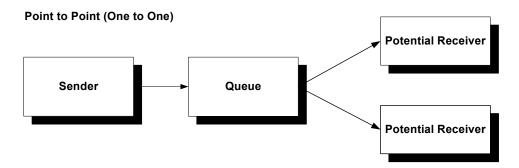


Point-to-Point Projects

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.

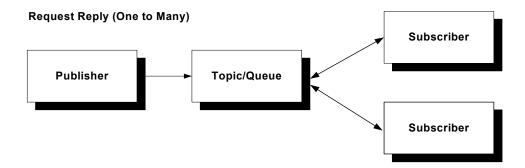
Figure 5 Point to Point



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 25
- XA Compliance on page 25
- Message Selectors on page 25
- Multi-Threaded Apartments on page 25
- Sample Code on page 26

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 156.

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:

- About Compensating Resource Managers on page 26
- Implementing the Compensating Resource Manager on page 28
- Configuring the Compensating Resource Manager on page 28

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 160.

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.

- Step 1: Add a New Component Application on page 28
- Step 2: Installing the stc_mscom Component on page 31
- Step 3: Configuring the stc_mscom Component on page 33
- Step 4: Configuring the STC_MSCOM.Compensator on page 33
- Step 5: Configuring the STC_MSCOM XA Connection Factories on page 35

Note: Before beginning the following procedure, make sure the stc_mscom.dll file is registered, as described in "Post-Installation Instructions" on page 17.

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.

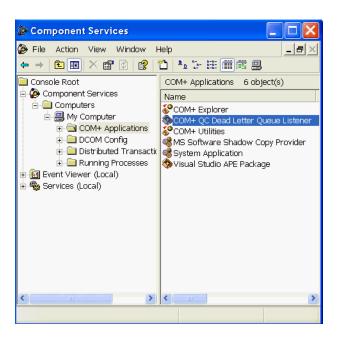


Figure 7 Component Services Folder

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

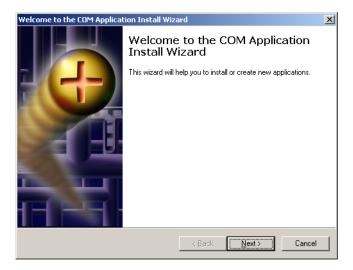


Figure 8 COM+ Application Install Wizard

- 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



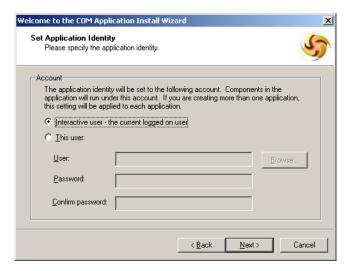
6 On the **Create Empty Application** window, enter the name **stc_mscom**, click the option button next to **Server application**, and then click **Next**.

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.

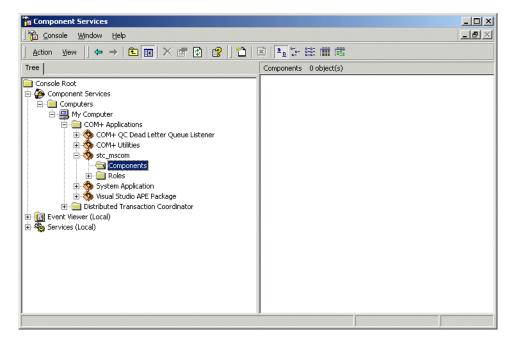
Step 2: Installing the stc_mscom Component

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 16).

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**.

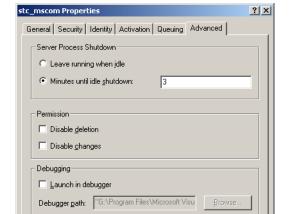


Figure 15 stc_mscom Properties: Advanced

Step 4: Configuring the STC_MSCOM.Compensator

▼ Enable Compensating Resource <u>Managers</u>

Enable 3GB support

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

OK

Cancel

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**.

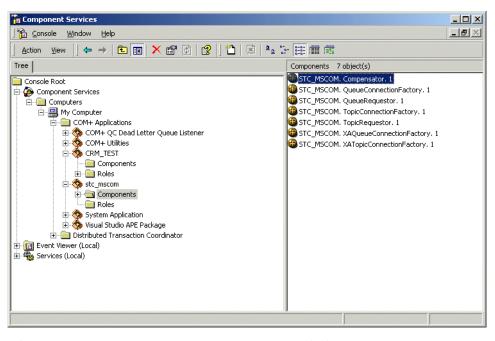
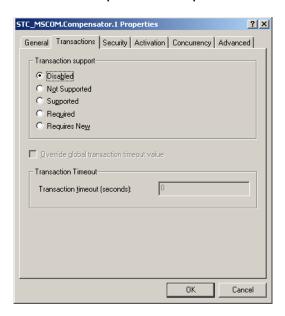


Figure 16 STC_MSCOM.Compensator Properties

The STC_MSCOM.Compensator Properties dialog appears.

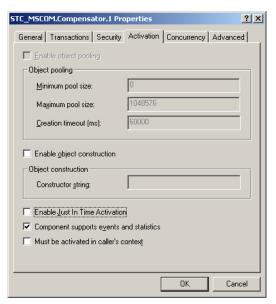
3 Click the Transactions tab, and then select **Disabled** in the **Transaction support** pane.

Figure 17 STC_MSCOM.Compensator Properties: Transaction Support



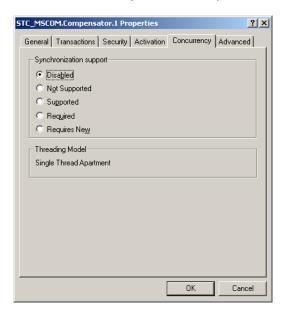
4 Click the Activation tab, and then deselect the **Enable Just In Time Activation** check box.

Figure 18 STC_MSCOM.Compenstator Properties: Activation



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

Figure 19 STC_MSCOM.Compensator Properties: Concurrency



6 Click OK.

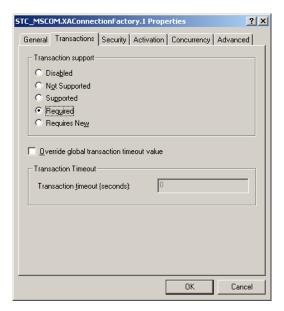
Step 5: Configuring the STC_MSCOM XA Connection Factories

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 the **Required** option 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 the **Required** option 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.

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

- About the Java CAPS JMS Interface for Applications in COM+ on page 38
- Viewing the COM+ API on page 41
- Creating Destinations on page 42
- COM+ Constants on page 42
- BytesMessage Object on page 44
- Connection Object on page 57
- ConnectionFactory Object on page 58
- ConnectionMetaData Object on page 59
- MapMessage Object on page 59
- message Object on page 72
- MessageConsumer Object on page 76
- MessageListener Object on page 77
- Queue Object on page 78
- QueueBrowser Object on page 79
- QueueConnection Object on page 79
- QueueConnectionFactory Object on page 81
- QueueReceiver Object on page 82
- QueueRequestor Object on page 84
- QueueSender Object on page 86
- QueueSession Object on page 88
- session Object on page 93
- StreamMessage Object on page 97
- TemporaryQueue Object on page 109
- TemporaryTopic Object on page 110

- TextMessage Object on page 112
- Topic Object on page 115
- TopicConnection Object on page 116
- TopicConnectionFactory Object on page 118
- TopicPublisher Object on page 119
- TopicRequestor Object on page 121
- TopicSession Object on page 122
- TopicSubscriber Object on page 129
- XAQueueConnection Object on page 131
- XAQueueConnectionFactory Object on page 134
- XASession Object on page 140
- XATopicConnection Object on page 144
- XATopicConnectionFactory Object on page 146
- XATopicSession Object on page 149
- Error Codes on page 153

4.1 About the Java CAPS JMS Interface for Applications in COM+

This section provides information about the COM+ API for the JMS interface, including wrapper functions and mapping to the JMS API. The full COM+ API reference begins with "BytesMessage Object" on page 44.

4.1.1 The COM+ Object Model

The eGate API Kit provides the **stc_mscom.dll** as its COM+ interface to access the Java CAPS JMS from COM+ applications. **Figure 21 on page 39** illustrates the Java CAPS JMS COM+ object model.

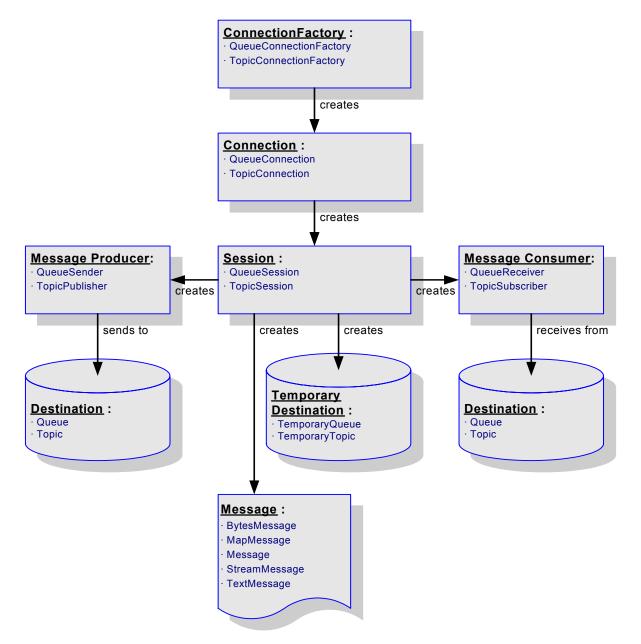


Figure 21 COM+ Object Model

4.1.2 Wrapper Functions for COM+ Applications

The API Kit supplies a set of JMS wrapper functions for COM+ applications to allow your applications to quickly access the Java CAPS JMS without your having to understand all JMS details.

While the wrapper functions are sufficient for most applications, they do not provide a complete function set; for details on the complete COM+ API, refer to the API descriptions in this chapter.

At a higher level of abstraction, you only need to manage a few types of structures.

- Session—Characterized by a hostname, port number, session type (either a pub/sub "topic session" or a point-to-point "queue session"), and connection parameters such as acknowledgment mode, transaction mode, delivery mode, and client ID.
- **Destination**—Either a topic (for pub/sub messaging) or else a queue (for point-topoint messaging). Characterized by session and destination name.
- Message Producer—Either a topic publisher or a queue sender. Characterized by session and destination.
- Message Consumer—Either a topic subscriber or a queue receiver. Characterized by session and destination (and, in pub/sub messaging only, by the name of the durable subscriber, if designated).
- Requestor—In request/reply messaging, either a topic requestor or a queue requestor. Characterized by session, destination, message, and time-to-live value expressed in milliseconds.
- **Message**—Characterized by the message type and payload. The payload is also called the message *body*, or message *content*.
 - For BytesMessage messages, the payload is an array of bytes.
 - For MapMessage messages, the payload is an array of name and value pairs.
 - For StreamMessage messages, the payload is a stream of primitive values.
 - For TextMessage messages, the payload is a string of text characters. Native encoding is used.

4.1.3 COM+ API Correspondence with JMS

eGate supports the Java Message Service (JMS) COM+ APIs listed in the API reference in this chapter. As much as possible, the COM+ API was designed to correspond to the standard JMS Java API. For example, the **message** and **BytesMessage** interfaces in the C++ API provide similar functionality to the JMS **Message** and **BytesMessage** interfaces. 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 JMS 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

Both APIs support the following features:

Publish/subscribe, point-to-point, and request/reply messaging

- Message selectors
- Local transactions
- Distributed transactions
- Session recovery
- Temporary topics and queues
- Acknowledgement modes

Differences between the two APIs include the following.

- The COM+ API does not support the ObjectMessage interface.
- The COM+ API does not support queue browsers or connection consumers.

Table 2 lists the differences between COM+ data types and Java data types.

Table 2 COM+ Type Mapping to Java Types

COM+ Data Type	Java Data Type
currency	long
long	int
unsigned char	byte
unsigned char *	bytes[]
float	float
double	double
String	String
int	short
short	char

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 156).

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 Creating Destinations

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

4.4 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 154.

4.4.1 AcknowledgeMode Constants

Table 3 lists the acknowledgement modes along with their corresponding integer values.

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.

4.4.2 DeliveryMode Constants

Table 4 lists the delivery mode constants and their corresponding integer values.

Table 4 Values for DeliveryMode Constants

Name	Value	Description
msNonPersistent	0	O indicates non-persistent delivery mode. 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 nonpersistent message with a one-time-only delivery. If the JMS IQ Manager goes down, the message is lost. This mode maximizes performance and should be used if an occasional lost message is tolerable.
msPersistent	1	1 indicates persistent delivery mode. This mode instructs the JMS IQ Manager to log the message to storage as part of the client's send operation. It maximizes reliability and should be used if the application will have problems if the message is lost in transit.

4.4.3 Message Constants

Table 5 lists the default values for the message delivery mode, priority, and expiration period.

 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.

4.4.4 Priority Constants

Priority constants define a ten-level priority system for messages (see "msDefaultPriority" in Table 5). Table 6 lists the priority constants and their corresponding integer values.

 Table 6
 Priority Level Constants

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

4.5 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.

4.5.1 BytesMessage Properties

The **BytesMessage** object includes the properties listed in Table 7.

 Table 7
 Properties of the BytesMessage Object

Property	Data Type	Description
CorrelationID	String	Sets or returns correlation ID values that are either JMS IQ Manager message IDs or application-specific strings.
CorrelationIDAsBytes	Variant	This is not currently supported.
DeliveryMode	BytesMessageConstant	Sets or returns the delivery mode for this message as either msNonPersistent, or msPersistent. The default value is msDefaultDeliveryMode (msPersistent). See DeliveryMode Constants on page 43.
Destination	Destination	This is not currently supported.
Expiration	Currency	Sets or returns the message expiration time in milliseconds.
MessageID	String	Sets or returns the value of the uniquely assigned identifier in the message header.
Priority	PriorityConstants	This is not currently supported.
Redelivered	Boolean	Sets or returns an indicator of whether the message is redelivered. A value of true indicates the message is redelivered; a value of false indicates it is not.
ReplyTo	Destination	Sets or returns the destination to which a reply to the message is sent. A destination can be a Topic, Queue, Temporary Topic, or a Temporary Queue.
Timestamp	Currency	Sets or returns the timestamp of the message.
Туре	String	Sets or returns the message type.

4.5.2 BytesMessage Methods

The **BytesMessage** object includes the following methods:

- Acknowledge on page 46
- ClearBody on page 46
- ClearProperties on page 47
- **GetProperty** on page 47

- ReadUnsignedShort on page 51
- **ReadUTF** on page 52
- Reset on page 52
- **SetProperty** on page 53

- PropertyExists on page 47
- ReadBoolean on page 48
- ReadByte on page 48
- ReadBytes on page 49
- ReadChar on page 49
- ReadDouble on page 49
- ReadFloat on page 50
- ReadInt on page 50
- ReadLong on page 50
- ReadShort on page 51
- ReadUnsignedByte on page 51

- WriteBoolean on page 53
- WriteByte on page 53
- WriteBytes on page 54
- WriteChar on page 54
- WriteDouble on page 55
- WriteFloat on page 55
- WriteInt on page 55
- WriteLong on page 56
- WriteObject on page 56
- WriteShort on page 56
- WriteUTF on page 57

Acknowledge

Syntax

Sub Acknowledge()

Description

Acknowledges the receipt of current and previous messages.

Parameters

None.

Return Value

A long value representing the result status. A value of 0 (zero) indicates the function was successful; any other value indicates there was an error.

ClearBody

Syntax

Sub ClearBody()

Description

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

Parameters

None.

Return Value

A long value representing the result status. A value of 0 (zero) indicates the function was successful; any other value indicates there was an error.

ClearProperties

Syntax

Sub ClearProperties()

Description

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

Parameters

None.

Return Value

A long value representing the result status. A value of 0 (zero) indicates the function was successful; any other value indicates there was an error.

GetProperty

Syntax

Function GetProperty(name As String)

Description

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

Parameters

Name	Туре	Description
name	String	The name of the property.

Return Value

The value of the specified property.

This method also returns a long value representing the result status. A value of 0 (zero) indicates the function was successful; any other value indicates there was an error.

PropertyExists

Syntax

Function PropertyExists (name As String) As Boolean

Description

Checks whether a value for a specific property exists.

Parameters

Name	Туре	Description	
name	String	The name of the property to check.	

Return Value

A Boolean indicator of whether a value exists for the specified property. If the property value exists, the return value is **true**; otherwise, it is **false**.

This method also returns a long value representing the result status. A value of 0 (zero) indicates the function was successful; any other value indicates there was an error.

ReadBoolean

Syntax

Function ReadBoolean() As Boolean

Description

Reads a Boolean value from the bytes message stream.

Parameters

None.

Return Value

The value of the Boolean property.

This method also returns a long value representing the result status. A value of 0 (zero) indicates the function was successful; any other value indicates there was an error.

ReadByte

Syntax

Function ReadByte() As Byte

Description

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

Parameters

None.

Return Value

The value of the Byte property.

This method also returns a long value representing the result status. A value of 0 (zero) indicates the function was successful; any other value indicates there was an error.

ReadBytes

Syntax

Function ReadBytes(value, [length]) As Long

Description

Reads a portion of the bytes message stream.

Parameters

Name	Туре	Description
value	Variant	The buffer into which the data is read.
length	Variant	The number of bytes to read.

Return Value

A long value representing the result status. A value of 0 (zero) indicates the function was successful; any other value indicates there was an error.

ReadChar

Syntax

Function ReadChar() As Short

Description

Reads a Unicode character value from the bytes message stream.

Parameter

None.

Return Value

The value of the character.

A long value representing the result status. A value of 0 (zero) indicates the function was successful; any other value indicates there was an error.

ReadDouble

Syntax

Function ReadDouble() As Double

Description

Reads a double value from the bytes message stream.

Parameters

None.

Return Value

The value of the double property.

This method also returns a long value representing the result status. A value of 0 (zero) indicates the function was successful; any other value indicates there was an error.

ReadFloat

Syntax

```
Function ReadFloat() As Single
```

Description

Reads a floating point value from the bytes message stream.

Parameters

None.

Return Value

The value of the floating point property.

This method also returns a long value representing the result status. A value of 0 (zero) indicates the function was successful; any other value indicates there was an error.

ReadInt

Syntax

```
Function ReadInt() As Long
```

Description

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

Parameters

None.

Return Value

The value of the integer property.

This method also returns a long value representing the result status. A value of 0 (zero) indicates the function was successful; any other value indicates there was an error.

ReadLong

Syntax

```
Function ReadLong() As Currency
```

Description

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

Parameters

None.

Return Value

The value of the long property.

This method also returns a long value representing the result status. A value of 0 (zero) indicates the function was successful; any other value indicates there was an error.

ReadShort

Syntax

```
Function ReadShort() As Integer
```

Description

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

Parameters

None.

Return Value

The value of the short property.

This method also returns a long value representing the result status. A value of 0 (zero) indicates the function was successful; any other value indicates there was an error.

ReadUnsignedByte

Syntax

```
Function ReadUnsignedByte() As Long
```

Description

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

Parameters

None.

Return Value

The value of the unsigned byte property.

This method also returns a long value representing the result status. A value of 0 (zero) indicates the function was successful; any other value indicates there was an error.

ReadUnsignedShort

Syntax

Function ReadUnsignedShort() As Long

Description

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

Parameters

None.

Return Value

The value of the unsigned short property.

This method also returns a long value representing the result status. A value of 0 (zero) indicates the function was successful; any other value indicates there was an error.

ReadUTF

Syntax

```
Function ReadUTF() As String
```

Description

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

Parameters

None.

Return Value

A String containing the value of the UTF-encoded property.

This method also returns a long value representing the result status. A value of 0 (zero) indicates the function was successful; any other value indicates there was an error.

Reset

Syntax

```
Sub Reset()
```

Description

Places the message body in read-only mode and repositions the stream of bytes to the beginning.

Parameters

None.

Return Value

None.

This method also returns a long value representing the result status. A value of 0 (zero) indicates the function was successful; any other value indicates there was an error.

SetProperty

Syntax

Sub SetProperty(name As String, value)

Description

Sets a Visual Basic data type value into the message property with the specified name.

Parameters

Name	Туре	Description
name	String	The name of the property.
value	Variant	The value to set in the specified property.

Return Value

None.

WriteBoolean

Syntax

Sub WriteBoolean(value As Boolean)

Description

Writes a Boolean value to the bytes message stream as a 1-byte value.

Parameters

Name	Туре	Description
value	Boolean	The value to write. The value true is written out as the value (byte)1; the value false is written out as the value (byte)0.

Return Value

A long value representing the result status. A value of 0 (zero) indicates the function was successful; any other value indicates there was an error.

WriteByte

Syntax

Sub WriteByte(value As Byte)

Description

Writes a 1-byte value to the bytes message stream.

Parameters

Name	Туре	Description
value	Byte	The value to write.

Return Value

A long value representing the result status. A value of 0 (zero) indicates the function was successful; any other value indicates there was an error.

WriteBytes

Syntax

```
Sub WriteBytes(value, [offset], [length])
```

Description

Writes a byte array, or a portion of the byte array, to the bytes message stream.

Parameters

Name	Туре	Description
value	Variant	The byte array value to write.
offset	Variant	The initial offset within the byte array.
length	Variant	The number of bytes to use.

Return Value

A long value representing the result status. A value of 0 (zero) indicates the function was successful; any other value indicates there was an error.

WriteChar

Syntax

Sub WriteChar(value As Short)

Description

Writes a char value to the bytes message stream as a 2-byte value, high byte first.

Parameters

Name	Туре	Description
value	Short	The value to write.

Return Value

A long value representing the result status. A value of 0 (zero) indicates the function was successful; any other value indicates there was an error.

WriteDouble

Syntax

Sub WriteDouble(value As Double)

Description

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

Parameters

Name	Туре	Description
value	Double	The value to write.

Return Value

A long value representing the result status. A value of 0 (zero) indicates the function was successful; any other value indicates there was an error.

WriteFloat

Syntax

Sub WriteFloat(Value As Single)

Description

Converts the floating point argument to a long value, and then writes a 4-byte long value to the bytes message stream (high byte is written first).

Parameters

Name	Туре	Description
value	Single	The value to write.

Return Value

A long value representing the result status. A value of 0 (zero) indicates the function was successful; any other value indicates there was an error.

WriteInt

Syntax

Sub WriteInt(value As Long)

Description

Writes a number to the bytes message stream as a 4-byte integer (high byte is written first).

Parameters

Name	Туре	Description
value	Long	The value to write.

Return Value

A long value representing the result status. A value of 0 (zero) indicates the function was successful; any other value indicates there was an error.

WriteLong

Syntax

Sub WriteLong(value As Currency)

Description

Writes a number to the bytes message stream as an 8-byte long value (high byte is written first).

Parameters

Name	Туре	Description
value	Currency	The value to write.

Return Value

A long value representing the result status. A value of 0 (zero) indicates the function was successful; any other value indicates there was an error.

WriteObject

This method is not currently supported.

WriteShort

Syntax

Sub WriteShort(value As Integer)

Description

Writes a number to the bytes message stream as a 2-byte short integer (high byte is written first).

Parameters

Name	Туре	Description
value	Integer	The value to write.

Return Value

A long value representing the result status. A value of 0 (zero) indicates the function was successful; any other value indicates there was an error.

WriteUTF

Syntax

Sub WriteUTF(value As String)

Description

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

Parameters

Name	Туре	Description
value	String	The value to write.

Return Value

A long value representing the result status. A value of 0 (zero) indicates the function was successful; any other value indicates there was an error.

4.6 Connection Object

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

4.6.1 Connection Properties

The **Connection** object includes the properties listed in Table 8.

Table 8 Properties of the Connection Object

Property	Data Type	Description
ClientID	String	Sets or returns the client identifier for this connection. This value is specified to the JMS IQ Manager.
MetaData	ConnectionMetaData	This is not currently supported.

4.6.2 Connection Methods

The **Connection** object includes the following methods:

- Start on page 58
- Stop on page 58

Start

Syntax

Sub Start()

Description

Starts or restarts the delivery of a transaction connection's incoming messages.

Parameters

None.

Return Value

A long value representing the result status. A value of 0 (zero) indicates the function was successful; any other value indicates there was an error.

Stop

Syntax

Sub Stop()

Description

Temporarily stops the delivery of incoming messages from a transaction connection.

Parameters

None.

Return Value

A long value representing the result status. A value of 0 (zero) indicates the function was successful; any other value indicates there was an error.

4.7 ConnectionFactory Object

A **ConnectionFactory** encapsulates a set of connection configuration parameters that is defined by the administrator. This is an abstract interface.

There are no methods associated with this object.

4.7.1 ConnectionFactory Properties

The **ConnectionFactory** object includes the properties listed in Table 9.

Table 9 Properties of the ConnectionFactory Object

Property	Data Type	Description
HostName	String	Sets or returns the name of the host where the message service is running.

 Table 9
 Properties of the ConnectionFactory Object

Property	Data Type	Description
Port	Long	Sets or returns the port number at which the message service is listening. The default value is 24053 .
PortOffset	Long	Sets or returns the port offset number of the message service if more then one service is running on the same host machine and using the same port number.

4.8 ConnectionMetaData Object

This object is currently not supported.

4.9 MapMessage Object

The **MapMessage** object 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.

4.9.1 MapMessage Properties

The **MapMessage** object includes the properties listed in Table 10.

 Table 10
 Properties of the MapMessage Object

Property	Data Type	Description
CorrelationID	String	Sets or returns correlation ID values that are either JMS IQ Manager message IDs or application-specific strings.
CorrelationIDAsBytes	Variant	This is not currently supported.
DeliveryMode	DeliveryModeConstant	Sets or returns the delivery mode for this message as either msNonPersistent, or msPersistent. The default value is msDefaultDeliveryMode (msPersistent). See DeliveryMode Constants on page 43.
Destination	Destination	This is not currently supported.
Expiration	Currency	Sets or returns the message expiration time in milliseconds.

 Table 10
 Properties of the MapMessage Object

Property	Data Type	Description
MapNames	Variant	Returns the map message names as an array of Strings. This property is read-only.
MessageID	String	Sets or returns the value of the uniquely assigned identifier in the message header.
Priority	PriorityConstants	This is not currently supported.
Redelivered	Boolean	Sets or returns an indicator of whether the message is redelivered. A value of true indicates the message is redelivered; a value of false indicates it is not.
ReplyTo	Destination	Sets or returns the destination to which a reply to the message is sent. A destination can be a Topic, Queue, TemporaryTopic, or a TemporaryQueue.
Timestamp	Currency	Sets or returns the message timestamp.
Туре	String	Sets or returns the message type.

4.9.2 MapMessage Object Methods

The **MapMessage** object includes the following methods:

- Acknowledge on page 61
- ClearBody on page 61
- ClearProperties on page 61
- GetBoolean on page 62
- GetByte on page 62
- GetBytes on page 62
- GetChar on page 63
- GetDouble on page 63
- GetFloat on page 64
- GetInt on page 64
- GetLong on page 65
- GetObject on page 65
- GetProperty on page 65
- GetShort on page 66
- GetString on page 66

- ItemExists on page 66
- PropertyExists on page 67
- SetBoolean on page 67
- **SetByte** on page 68
- SetBytes on page 68
- SetChar on page 69
- **SetDouble** on page 69
- SetFloat on page 69
- SetInt on page 70
- SetLong on page 70
- SetObject on page 71
- SetProperty on page 71
- SetShort on page 71
- SetString on page 72

Acknowledge

Syntax

Sub Acknowledge()

Description

Acknowledges the receipt of current and previous messages.

Parameters

None.

Return Value

A long value representing the result status. A value of 0 (zero) indicates the function was successful; any other value indicates there was an error.

ClearBody

Syntax

```
Sub ClearBody()
```

Description

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

Parameters

None.

Return Value

A long value representing the result status. A value of 0 (zero) indicates the function was successful; any other value indicates there was an error.

ClearProperties

Syntax

```
Sub ClearProperties()
```

Description

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

Parameters

None.

Return Value

A long value representing the result status. A value of 0 (zero) indicates the function was successful; any other value indicates there was an error.

GetBoolean

Syntax

Function GetBoolean (name As String) As Boolean

Description

Retrieves the value of the specified Boolean property.

Parameters

Name	Туре	Description
name	String	The name of the property to check.

Return Value

The value of the Boolean property.

This method also returns a long value representing the result status. A value of 0 (zero) indicates the function was successful; any other value indicates there was an error.

GetByte

Syntax

Function GetByte(name As String) As Byte

Description

Retrieves the value of the specified byte property.

Parameters

Name	Туре	Description
name	String	The name of the property to check.

Return Value

The value of the byte property.

This method also returns a long value representing the result status. A value of 0 (zero) indicates the function was successful; any other value indicates there was an error.

GetBytes

Syntax

Function GetBytes (name As String, length As Long)

Description

Retrieves a byte array value from the specified byte property.

Parameters

Name	Туре	Description
name	String	The name of the property to check.
length	Long	The number of bytes to retrieve.

Return Value

A byte array value from the byte property.

This method also returns a long value representing the result status. A value of 0 (zero) indicates the function was successful; any other value indicates there was an error.

GetChar

Syntax

Function GetChar(name As String) As Short

Description

Retrieves the Unicode character value of the specified property.

Parameters

Name	Туре	Description
name	String	The name of the property to check.

Return Value

The value of the specified property.

This method also returns a long value representing the result status. A value of 0 (zero) indicates the function was successful; any other value indicates there was an error.

GetDouble

Syntax

Function GetDouble(name As String) As Double

Description

Retrieves the value of the specified double property.

Parameters

Name	Туре	Description
name	String	The name of the property to check.

Return Value

The value of the double property.

This method also returns a long value representing the result status. A value of 0 (zero) indicates the function was successful; any other value indicates there was an error.

GetFloat

Syntax

Function GetFloat (name As String) As Single

Description

Retrieves the Single value of the specified floating point property.

Parameters

Name	Туре	Description
name	String	The name of the property to check.

Return Value

The value of the floating point property as a Single value.

This method also returns a long value representing the result status. A value of 0 (zero) indicates the function was successful; any other value indicates there was an error.

GetInt

Syntax

Function GetInt(name as a String) As Long

Description

Retrieves the long value of the specified integer property.

Parameters

Name	Туре	Description
name	String	The name of the property to check.

Return Value

The value of the integer property as a Long value.

This method also returns a long value representing the result status. A value of 0 (zero) indicates the function was successful; any other value indicates there was an error.

GetLong

Syntax

Function GetLong(name As String) As Currency

Description

Retrieves the Currency value of the specified long property.

Parameters

Name	Туре	Description
name	String	The name of the property to check.

Return Value

The value of the long property as a Currency value.

This method also returns a long value representing the result status. A value of 0 (zero) indicates the function was successful; any other value indicates there was an error.

GetObject

This method is not currently supported.

GetProperty

Syntax

Function GetProperty(name As String)

Description

Retrieves the Visual Basic data type property value with the given name into the MapMessage.

Parameters

Name	Туре	Description
name	String	The name of the property to check.

Return Value

The value of the specified property.

This method also returns a long value representing the result status. A value of 0 (zero) indicates the function was successful; any other value indicates there was an error.

GetShort

Syntax

Function GetShort(name As String) As Integer

Description

Retrieves the Integer value of the specified short property.

Parameters

Name	Туре	Description
name	String	The name of the property to check.

Return Value

The value of the short property as an Integer value.

This method also returns a long value representing the result status. A value of 0 (zero) indicates the function was successful; any other value indicates there was an error.

GetString

Syntax

Function GetString (name As String) As String

Description

Retrieves the value of the specified String property.

Parameters

Name	Туре	Description
name	String	The name of the property to check.

Return Value

The value of the String property.

This method also returns a long value representing the result status. A value of 0 (zero) indicates the function was successful; any other value indicates there was an error.

ItemExists

Syntax

Function ItemExists(name As String) As Boolean

Description

Checks whether the specified item exists in the MapMessage.

Parameters

Name	Туре	Description
name	String	The name of the item to check.

Return Value

A Boolean indicator of whether the item exists. If the item exists, the return value is **true**; otherwise it is **false**.

This method also returns a long value representing the result status. A value of 0 (zero) indicates the function was successful; any other value indicates there was an error.

PropertyExists

Syntax

Function PropertyExists (name As String) As Boolean

Description

Checks whether a value for the specified property exists.

Parameters

Name	Туре	Description
name	String	The name of the property to check.

Return Value

A Boolean indicator of whether the property value exists. If the value exists, the return value is **true**; otherwise it is **false**.

This method also returns a long value representing the result status. A value of 0 (zero) indicates the function was successful; any other value indicates there was an error.

SetBoolean

Syntax

Sub SetBoolean (name As String, value As Boolean)

Description

Sets a Boolean property value with the specified name into the MapMessage.

Parameters

Name	Туре	Description
name	String	The name of the property to update.
value	Boolean	The Boolean value to set.

Return Value

A long value representing the result status. A value of 0 (zero) indicates the function was successful; any other value indicates there was an error.

SetByte

Syntax

Sub SetByte(name As String, value As Byte)

Description

Sets a byte property value with the specified name into the MapMessage.

Parameters

Name	Туре	Description
name	String	The name of the property to update.
value	Byte	The byte value to set.

Return Value

A long value representing the result status. A value of 0 (zero) indicates the function was successful; any other value indicates there was an error.

SetBytes

Syntax

Sub SetBytes(name As String, value, [offset], [length])

Description

Sets a byte array or a portion of the value with the specified name into the MapMessage.

Parameters

Name	Туре	Description
name	String	The name of the property to update.
value	Variant	The value to set.
offset	Variant	The initial offset within the byte array.
length	Variant	The number of bytes to use.

Return Value

A long value representing the result status. A value of 0 (zero) indicates the function was successful; any other value indicates there was an error.

SetChar

Syntax

Sub SetChar(name As String, value As Short)

Description

Sets a Unicode character value with the specified name into the MapMessage.

Parameters

Name	Туре	Description
name	String	The name of the property to update.
value	Short	The value to set.

Return Value

A long value representing the result status. A value of 0 (zero) indicates the function was successful; any other value indicates there was an error.

SetDouble

Syntax

Sub SetDouble (name As String, value As Double)

Description

Sets a double property value with the specified name into the MapMessage.

Parameters

Name	Туре	Description
name	String	The name of the property to update.
value	Double	The Double value to set.

Return Value

A long value representing the result status. A value of 0 (zero) indicates the function was successful; any other value indicates there was an error.

SetFloat

Syntax

Sub SetFloat (name As String, value As Single)

Description

Sets a floating point property value with the specified name into the MapMessage.

Parameters

Name	Туре	Description
name	String	The name of the property to update.
value	Single	The floating point value to set.

Return Value

A long value representing the result status. A value of 0 (zero) indicates the function was successful; any other value indicates there was an error.

SetInt

Syntax

Sub SetInt(name As String, value As Long)

Description

Sets an integer property value with the specified name into the MapMessage.

Parameters

Name	Туре	Description
name	String	The name of the property to update.
value	Long	The value to set.

Return Value

A long value representing the result status. A value of 0 (zero) indicates the function was successful; any other value indicates there was an error.

SetLong

Syntax

Sub SetLong(name As String, value As Currency)

Description

Sets a long property value with the specified name into the MapMessage.

Parameters

Name	Туре	Description
name	String	The name of the property to update.
value	Currency	The long value to set.

Return Value

A long value representing the result status. A value of 0 (zero) indicates the function was successful; any other value indicates there was an error.

SetObject

This method is not currently supported.

SetProperty

Syntax

Sub SetProperty(name As String, value)

Description

Sets a Visual Basic data type property value with the specified name into the MapMessage.

Parameters

Name	Туре	Description
name	String	The name of the property to update.
value	Variant	The value to set.

Return Value

A long value representing the result status. A value of 0 (zero) indicates the function was successful; any other value indicates there was an error.

SetShort

Syntax

Sub SetShort(name As String, value As Integer)

Description

Sets a short property value with the specified name into the MapMessage.

Parameters

Name	Туре	Description
name	String	The name of the property to update.
value	Integer	The value to set.

Return Value

A long value representing the result status. A value of 0 (zero) indicates the function was successful; any other value indicates there was an error.

SetString

Syntax

Sub SetString(name As String, value As String)

Description

Sets a String property value with the specified name into the MapMessage.

Parameters

Name	Туре	Description
name	String	The name of the property to update.
value	String	The String value to set.

Return Value

A long value representing the result status. A value of 0 (zero) indicates the function was successful; any other value indicates there was an error.

4.10 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**.

4.10.1 message Properties

The **message** object includes the properties listed in Table 11.

Table 11 Properties of the message Object

Property	Data Type	Description
CorrelationID	String	Sets or returns correlation ID values that are either JMS IQ Manager message IDs or application-specific strings.
CorrelationIDAsBytes	Variant	This is not currently supported.
DeliveryMode	DeliveryModeConstant	Sets or returns the delivery mode for this message as either msNonPersistent, or msPersistent. The default value is msDefaultDeliveryMode (msPersistent). See DeliveryMode Constants on page 43.
Destination	Destination	This is not currently supported.
Expiration	Currency	Sets or returns the message expiration time in milliseconds.

 Table 11
 Properties of the message Object

Property	Data Type	Description
MessageID	String	Sets or returns the value of the uniquely assigned identifier in the message header.
Priority	PriorityConstants	This is not currently supported.
Redelivered	Boolean	Sets or returns an indicator of whether the message is redelivered. A value of true indicates the message is redelivered; a value of false indicates it is not.
ReplyTo	Destination	Sets or returns the destination to which a reply to the message is sent. A destination can be a Topic, Queue, TemporaryTopic, or a TemporaryQueue.
Timestamp	Currency	Sets or returns the message timestamp.
Туре	String	Sets or returns the message type.

4.10.2 message Methods

The **message** object includes the following methods:

- Acknowledge on page 73
- ClearBody on page 74
- ClearProperties on page 74

- GetProperty on page 74
- PropertyExists on page 75
- SetProperty Method on page 75

Acknowledge

Syntax

Sub Acknowledge()

Description

Acknowledges the receipt of current and previous messages.

Parameters

None.

Return Value

A long value representing the result status. A value of 0 (zero) indicates the function was successful; any other value indicates there was an error.

ClearBody

Syntax

Sub ClearBody()

Description

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

Parameters

None.

Return Value

A long value representing the result status. A value of 0 (zero) indicates the function was successful; any other value indicates there was an error.

ClearProperties

Syntax

Sub ClearProperties()

Description

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

Parameters

None.

Return Value

A long value representing the result status. A value of 0 (zero) indicates the function was successful; any other value indicates there was an error.

GetProperty

Syntax

Function GetProperty(name As String)

Description

Retrieves the Visual Basic data type property value with the given name into the message.

Parameters

Name	Туре	Description
name	String	The name of the property to check.

The value of the specified property.

This method also returns a long value representing the result status. A value of 0 (zero) indicates the function was successful; any other value indicates there was an error.

PropertyExists

Syntax

Function PropertyExists (name As String) As Boolean

Description

Checks whether a value for the specified property exists.

Parameters

Name	Туре	Description
name	String	The name of the property to check.

Return Value

A Boolean indicator of whether the property value exists. If the value exists, the return value is **true**; otherwise it is **false**.

A long value representing the result status. A value of 0 (zero) indicates the function was successful; any other value indicates there was an error.

SetProperty Method

Syntax

Sub SetProperty(name As String, value)

Description

Sets a Visual Basic data type value into the message property with the specified name.

Parameters

Name	Туре	Description
name	String	The name of the property.
value	Variant	The value to set in the specified property.

Return Value

A long value representing the result status. A value of 0 (zero) indicates the function was successful; any other value indicates there was an error.

4.11 MessageConsumer Object

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

4.11.1 MessageConsumer Properties

The MessageConsumer object includes the properties listed in Table 12.

 Table 12
 Properties of the MessageConsumer Object

Property	Data Type	Description
MessageListener	MessageListener	This is not currently supported.
MessageSelector	String	Returns the message consumer's selector expression.

4.11.2 Message Consumer Methods

The **MessageConsumer** object includes the following methods:

- Close on page 76
- ReceiveMessage on page 76
- ReceiveNoWait on page 77

Close

Syntax

Sub Close()

Description

Closes resources on behalf of a message consumer. A message service can allocate resources on behalf of a message consumer, so it is recommended that you close any unused resources.

Parameters

None.

Return Value

A long value representing the result status. A value of 0 (zero) indicates the function was successful; any other value indicates there was an error.

ReceiveMessage

Syntax

Function Receive([timeOut]) As message

Description

Receives the next message produced or the next message that arrives within the specified timeout interval for the message consumer.

Parameters

Name	Туре	Description
timeOut	Variant	The timeout value of the message consumer in milliseconds.

Return Value

A long value representing the result status. A value of 0 (zero) indicates the function was successful; any other value indicates there was an error.

ReceiveNoWait

Syntax

Function ReceiveNoWait() As message

Description

Receives the next message if one is immediately available.

Parameters

None.

Return Value

A long value representing the result status. A value of 0 (zero) indicates the function was successful; any other value indicates there was an error.

4.12 MessageListener Object

This object is not currently supported.

4.13 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.

4.13.1 MessageProducer Properties

The MessageProducer object includes the properties listed in Table 13.

 Table 13
 Properties of the MessageProducer Object

Property	Data Type	Description
DeliveryMode	DeliveryModeConstant	Sets or returns the delivery mode for this message as either msNonPersistent, or msPersistent. The default value is msDefaultDeliveryMode (msPersistent). See DeliveryMode Constants on page 43.
DisableMessageID	Boolean	Sets or returns an indicator of whether message IDs are disabled. A value of true indicates message IDs are disabled; false indicates they are not.
DisableMessageTimestamp	Boolean	Sets or returns an indicator of whether timestamping is disabled. A value of true indicates timestamping is disabled; false indicates it is not.
Priority	PriorityConstants	This is not currently supported.
TimeToLive	Currency	Sets or returns 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 (0 (zero), which is unlimited).

4.14 Queue Object

A **Queue** object encapsulates a message service-specific queue name.

4.14.1 Queue Properties

The **Queue** object includes the property listed in Table 14.

 Table 14
 Properties of the Queue Object

Property	Data Type	Description
QueueName	String	Returns the name of this queue. This property is read-only.

4.14.2 Queue Methods

The **Queue** object includes the following method:

ToString on page 79

ToString

Syntax

Function ToString() As String

Description

Retrieves a printed version of the queue name.

Parameters

None.

Return Value

The queue name as a String.

This method also returns a long value representing the result status. A value of 0 (zero) indicates the function was successful; any other value indicates there was an error.

4.15 QueueBrowser Object

This object is currently not supported.

4.16 QueueConnection Object

A **QueueConnection** is an active connection to a point-to-point message service.

4.16.1 QueueConnection Properties

The **QueueConnection** object includes the properties listed in Table 15.

Table 15 Properties of the QueueConnection Object

Property	Data Type	Description
ClientID	String	Sets or returns the client identifier for the connection.
MetaData	ConnectionMetaData	This is not currently supported.

4.16.2 QueueConnection Methods

The **QueueConnection** object includes the following methods:

- CreateQueueSession on page 80
- Start on page 80
- Stop on page 81

CreateQueueSession

Syntax

Function CreateQueueSession(Transacted As Boolean, acknowledgeMode As AcknowledgeModeConstants) As QueueSession

Description

Creates a **QueueSession** using the specified transacted and acknowledge modes.

Parameters

Name	Туре	Description
Transacted	Boolean	An indicator of the whether the session is transacted. A value of true indicates the session is transacted; false means it is not.
acknowledgeMode	AcknowledgeModeConstants	An indicator of the acknowledgement mode. See AcknowledgeMode Constants on page 42 for possible values.

Return Value

A queue session.

This method also returns a long value representing the result status. A value of 0 (zero) indicates the function was successful; any other value indicates there was an error.

Start

Syntax

Sub Start()

Description

Starts or restarts a connection's delivery of incoming messages.

Parameters

None.

Return Value

A long value representing the result status. A value of 0 (zero) indicates the function was successful; any other value indicates there was an error.

Stop

Syntax

Sub Stop()

Description

Temporarily stops a connection's delivery of incoming messages.

Parameters

None.

Return Value

A long value representing the result status. A value of 0 (zero) indicates the function was successful; any other value indicates there was an error.

4.17 QueueConnectionFactory Object

A client uses a **QueueConnectionFactory** to create queue connections (**QueueConnection**) for a point-to-point message service.

4.17.1 QueueConnectionFactory Properties

The **QueueConnectionFactory** object includes the properties listed in Table 16. **Table 16** Properties of the QueueConnectionFactory Object

Property	Data Type	Description
HostName	String	Sets or returns the name of the host where the message service is running.
Port	Long	Sets or returns the port number at which the message service is listening. The default value is 24053 .
PortOffset	Long	Sets or returns the port offset number of the message service if more then one service is running on the same host machine and using the same port number.

4.17.2 QueueConnectionFactory Methods

The **QueueConnectionFactory** object includes the following methods:

- CreateQueueConnection on page 82
- CreateQueueConnectionEx on page 82

CreateQueueConnection

Syntax

Function CreateQueueConnection() As QueueConnection

Description

Creates a queue connection with a default user identity.

Parameters 4 8 1

None.

Return Value

A queue connection.

This method also returns a long value representing the result status. A value of 0 (zero) indicates the function was successful; any other value indicates there was an error.

CreateQueueConnectionEx

Syntax

Function CreateQueueConnection($userId\ As\ String$, $password\ As\ String$) As QueueConnection

Description

Creates a queue connection with the specified user identity.

Parameters

Name	Туре	Description
userId	String	The logon user ID.
password	String	The password associated with the specified user ID.

Return Value

A queue connection.

This method also returns a long value representing the result status. A value of 0 (zero) indicates the function was successful; any other value indicates there was an error.

4.18 QueueReceiver Object

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

4.18.1 QueueReceiver Properties

The **QueueReceiver** object includes the properties listed in Table 17.

 Table 17
 Properties of the QueueReceiver Object

Property	Data Type	Description
MessageListener	MessageListener	This is not currently supported.
MessageSelector	String	Returns the message selector expression for the message consumer. This property is read-only.
Queue	Queue	Returns the queue associated with the queue receiver.

4.18.2 QueueReceiver Methods

The QueueReceiver object includes the following methods:

- Close on page 83
- Receive on page 83
- ReceiveNoWait on page 84

Close

Syntax

Sub Close()

Description

Closes the queue receiver. A message service might allocate resources on behalf of a message consumer, so the receiver should be closed when it is no longer needed.

Parameters

None.

Return Value

A long value representing the result status. A value of 0 (zero) indicates the function was successful; any other value indicates there was an error.

Receive

Syntax

Function Receive([timeOut]) As message

Description

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

Parameters

Name	Туре	Description
timeOut	Variant	The timeout value for the message consumer in milliseconds.

Return Value

A message object.

This method also returns a long value representing the result status. A value of 0 (zero) indicates the function was successful; any other value indicates there was an error.

ReceiveNoWait

Syntax

Function ReceiveNoWait() As message

Description

Receives the next message if one is immediately available.

Parameters

None

Return Value

A message object.

This method also returns a long value representing the result status. A value of 0 (zero) indicates the function was successful; any other value indicates there was an error.

4.19 QueueRequestor Object

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

4.19.1 QueueRequestor Methods

The **QueueRequestor** object includes the following methods:

- Close on page 85
- Create on page 85
- Request on page 85

Close

Syntax

Sub Close()

Description

Closes the queue requestor.

Parameters

None.

Return Value

A long value representing the result status. A value of 0 (zero) indicates the function was successful; any other value indicates there was an error.

Create

Syntax

Sub Create(session As QueueSession, Queue As Queue)

Description

Creates a queue requestor.

Parameters

Name	Туре	Description
session	QueueSession	The queue session.
Queue	Queue	The queue associated with the queue requestor.

Return Value

A long value representing the result status. A value of 0 (zero) indicates the function was successful; any other value indicates there was an error.

Request

Syntax

Function Request(message As message, [timeOut]) As message

Description

Sends a request to the queue and waits for a reply, optionally for a specified amount of time.

Parameters

Name	Туре	Description
message	message	The message to send.
timeOut	Variant	The timeout value for the message in milliseconds.

Return Value

A reply message.

This method also returns a long value representing the result status. A value of 0 (zero) indicates the function was successful; any other value indicates there was an error.

4.20 QueueSender Object

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

4.20.1 QueueSender Properties

The **QueueSender** object includes the properties listed in Table 18.

 Table 18 Properties of the QueueSender Object

Property	Data Type	Description
DeliveryMode	DeliveryModeConstants	Sets or returns the delivery mode for this message as either msNonPersistent, or msPersistent. The default value is msDefaultDeliveryMode (msPersistent). See DeliveryMode Constants on page 43.
DisableMessageID	Boolean	Sets or returns an indicator of whether message IDs are disabled. A value of true indicates message IDs are disabled; false indicates they are not.
DisableMessageTimestamp	Boolean	Sets or returns an indicator of whether timestamping is disabled. A value of true indicates timestamping is disabled; false indicates it is not.
Priority	PriorityConstants	This is not currently supported.
Queue	Queue	Returns the queue associated with the queue sender. This property is read-only.

 Table 18
 Properties of the QueueSender Object

Property	Data Type	Description
TimeToLive	Currency	Sets or returns 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 (0 (zero), which is unlimited).

4.20.2 QueueSender Methods

The **QueueSender** object includes the following method:

Send on page 87

Send

Syntax

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

Description

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

Parameters

Name	Туре	Description
message	message	The message to send.
DeliveryMode	Variant	The delivery mode for the message. See DeliveryMode Constants on page 43.
Priority	Variant	The priority for the message (not currently supported).
TimeToLive	Variant	The lifetime of the message in milliseconds.
Queue	Variant	The queue to which the message should be sent.

Return Value

A long value representing the result status. A value of 0 (zero) indicates the function was successful; any other value indicates there was an error.

4.21 QueueSession Object

A **QueueSession** provides methods for creating the following objects: **QueueReceiver**, **QueueSender**, **QueueBrowser**, and **TemporaryQueue**.

4.21.1 QueueSession Properties

The **QueueSession** object includes the properties listed in Table 19.

 Table 19 Properties of the QueueSession Object

Property	Data Type	Description
MessageListener	MessageListener	This is not currently supported.
Transacted	Boolean	Returns an indicator of whether the session is in transacted mode. A value of true indicates the session is in transacted mode; a value of false indicates it is not. This property is readonly.

4.21.2 QueueSession Object Methods

The **QueueSession** object includes the following methods:

- Commit on page 88
- CreateBytesMessage on page 89
- CreateMapMessage on page 89
- CreateMessage on page 89
- **CreateQueue** on page 90
- CreateReceiver on page 90
- CreateSender on page 91

- CreateStreamMessage on page 91
- CreateTemporaryQueue on page 91
- CreateTextMessage on page 92
- Recover on page 92
- Rollback on page 92
- Run on page 93

Commit

Syntax

Sub Commit()

Description

Commits all messages processed in this transaction and releases any locks currently held.

Parameters

None.

A long value representing the result status. A value of 0 (zero) indicates the function was successful; any other value indicates there was an error.

CreateBytesMessage

Syntax

Function CreateBytesMessage() As BytesMessage

Description

Creates a BytesMessage object (see "BytesMessage Object" on page 44).

Parameters

None.

Return Value

A BytesMessage object.

This method also returns a long value representing the result status. A value of 0 (zero) indicates the function was successful; any other value indicates there was an error.

CreateMapMessage

Syntax

Function CreateMapMessage() As MapMessage

Description

Creates a MapMessage object (see "MapMessage Object" on page 59).

Parameters

None.

Return Value

A **MapMessage** object.

This method also returns a long value representing the result status. A value of 0 (zero) indicates the function was successful; any other value indicates there was an error.

CreateMessage

Syntax

Function CreateMessage() As message

Description

Creates a message object (see "message Object" on page 72).

Parameters

None.

A **message** object.

This method also returns a long value representing the result status. A value of 0 (zero) indicates the function was successful; any other value indicates there was an error.

CreateQueue

Syntax

Function CreateQueue(QueueName As String) As Queue

Description

Creates a queue identity given a queue name.

Parameters

Name	Туре	Description
QueueName	String	The name of the queue to create.

Return Value

A Queue object.

This method also returns a long value representing the result status. A value of 0 (zero) indicates the function was successful; any other value indicates there was an error.

CreateReceiver

Syntax

Function CreateReceiver(Queue As Queue, [MessageSelector]) As QueueReceiver

Description

Creates a **QueueReceiver** to receive messages for the specified queue.

Parameters

Name	Туре	Description
Queue	Queue	The queue to access.
MessageSelector	Variant	The message selector expression that a message's properties must match in order to be delivered.

Return Value

A **QueueReceiver** object.

This method also returns a long value representing the result status. A value of 0 (zero) indicates the function was successful; any other value indicates there was an error.

CreateSender

Syntax

Function CreateSender (Queue As Queue) As QueueSender

Description

Creates a **QueueSender** to send messages to the specified queue.

Parameters

Name	Туре	Description
Queue	Queue	The queue to which the messages will be sent.

Return Value

A QueueSender object.

This method also returns a long value representing the result status. A value of 0 (zero) indicates the function was successful; any other value indicates there was an error.

CreateStreamMessage

Syntax

Function CreateStreamMessage() As StreamMessage

Description

Creates a StreamMessage Object (see "StreamMessage Object" on page 97).

Parameters

None.

Return Value

A **StreamMessage** object.

This method also returns a long value representing the result status. A value of 0 (zero) indicates the function was successful; any other value indicates there was an error.

CreateTemporaryQueue

Syntax

Function CreateTemporaryQueue() As TemporaryQueue

Description

Creates a temporary queue.

Parameters

None.

A temporary queue.

This method also returns a long value representing the result status. A value of 0 (zero) indicates the function was successful; any other value indicates there was an error.

CreateTextMessage

Syntax

Function CreateTextMessage([Text]) As TextMessage

Description

Creates a TextMessage object (see "TextMessage Object" on page 112).

Parameters

Name	Туре	Description
Text	Variant	The string used to initialize the message.

Return Value

A **TextMessage** object.

This method also returns a long value representing the result status. A value of 0 (zero) indicates the function was successful; any other value indicates there was an error.

Recover

Syntax

```
Sub Recover()
```

Description

Stops message delivery in this session, and then restarts message delivery with the oldest unacknowledged message.

Parameters

None.

Return Value

A long value representing the result status. A value of 0 (zero) indicates the function was successful; any other value indicates there was an error.

Rollback

Syntax

Sub Rollback()

Description

Rolls back any messages processed in this transaction and releases any locks currently held.

Parameters

None.

Return Value

A long value representing the result status. A value of 0 (zero) indicates the function was successful; any other value indicates there was an error.

Run

Syntax

Sub Run()

Description

An optional operation that is only intended to be used by Application Servers and not by ordinary JMS clients.

Parameters

None.

Return Value

A long value representing the result status. A value of 0 (zero) indicates the function was successful; any other value indicates there was an error.

4.22 session Object

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

4.22.1 session Properties

The **session** object includes the properties listed in Table 20.

Table 20 Properties of the session Object

Property	Data Type	Description
MessageListener	MessageListener	This is not currently supported.
Transacted	Boolean	Returns an indicator of whether the session is in transacted mode. A value of true indicates the session is in transacted mode; a value of false indicates it is not. This property is readonly.

4.22.2 session Methods

The **session** object includes the following methods:

- Commit on page 94
- CreateBytesMessage on page 94
- CreateMapMessage on page 95
- CreateMessage on page 95
- CreateStreamMessage on page 95

- CreateTextMessage on page 96
- Recover on page 96
- Rollback on page 96
- Run on page 97

Commit

Syntax

Sub Commit()

Description

Commits all messages processed in this transaction and releases any locks currently held.

Parameters

None.

Return Value

A long value representing the result status. A value of 0 (zero) indicates the function was successful; any other value indicates there was an error.

Create Bytes Message

Syntax

Function CreateBytesMessage() As BytesMessage

Description

Creates a BytesMessage Object (see "BytesMessage Object" on page 44).

Parameters

None.

Return Value

A **BytesMessage** object.

This method also returns a long value representing the result status. A value of 0 (zero) indicates the function was successful; any other value indicates there was an error.

CreateMapMessage

Syntax

Function CreateMapMessage() As MapMessage

Description

Creates a MapMessage Object (see "MapMessage Object" on page 59).

Parameters

None.

Return Value

A MapMessage object.

This method also returns a long value representing the result status. A value of 0 (zero) indicates the function was successful; any other value indicates there was an error.

CreateMessage

Syntax

Function CreateMessage() As message

Description

Creates a message object (see "message Object" on page 72).

Parameters

None.

Return Value

A **message** object.

This method also returns a long value representing the result status. A value of 0 (zero) indicates the function was successful; any other value indicates there was an error.

Create Stream Message

Syntax

Function CreateStreamMessage() As StreamMessage

Description

Creates a **StreamMessage** object (see "**StreamMessage Object**" on page 97).

Parameters

None.

Return Value

A **StreamMessage** object.

This method also returns a long value representing the result status. A value of 0 (zero) indicates the function was successful; any other value indicates there was an error.

CreateTextMessage

Syntax

Function CreateTextMessage([Text]) As TextMessage

Description

Creates a TextMessage object (see "TextMessage Object" on page 112).

Parameters

Name	Туре	Description
Text	Variant	The string used to initialize the message.

Return Value

A **TextMessage** object.

This method also returns a long value representing the result status. A value of 0 (zero) indicates the function was successful; any other value indicates there was an error.

Recover

Syntax

Sub Recover()

Description

Stops message delivery for the session, and then restarts message delivery starting with the oldest unacknowledged message.

Parameters

None.

Return Value

A long value representing the result status. A value of 0 (zero) indicates the function was successful; any other value indicates there was an error.

Rollback

Syntax

Sub Rollback()

Description

Rolls back any messages processed in this transaction and releases any locks currently held.

Parameters

None.

Return Value

A long value representing the result status. A value of 0 (zero) indicates the function was successful; any other value indicates there was an error.

Run

Syntax

Sub Run()

Description

An optional operation that is only intended to be used by Application Servers and not by ordinary JMS clients.

Parameters

None.

Return Value

A long value representing the result status. A value of 0 (zero) indicates the function was successful; any other value indicates there was an error.

4.23 StreamMessage Object

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

4.23.1 StreamMessage Properties

The StreamMessage object includes the properties listed in Table 21.

 Table 21
 Properties of the StreamMessage Object

Property	Data Type	Description
CorrelationID	String	Sets or returns correlation ID values that are either JMS IQ Manager message IDs or application-specific strings.
CorrelationIDAsBytes	Variant	Sets or returns the correlation ID as an array of bytes for the message.
DeliveryMode	DeliveryModeConstants	Sets or returns the delivery mode for this message as either msNonPersistent, or msPersistent. The default value is msDefaultDeliveryMode (msPersistent). See DeliveryMode Constants on page 43.

 Table 21
 Properties of the StreamMessage Object

Property	Data Type	Description
Destination	Destination	Sets or returns the destination for the message.
Expiration	Currency	Sets or returns the message expiration time in milliseconds.
MessageID	String	Sets or returns the value of the uniquely assigned identifier in the message header.
Priority	PriorityConstants	Sets or returns the priority assigned to the message. Possible values are 1 - 9.
Redelivered	Boolean	Sets or returns an indicator of whether the message is redelivered. A value of true indicates the message is redelivered; a value of false indicates it is not.
ReplyTo	Destination	Sets or returns the destination to which a reply to the message is sent. A destination can be a Topic, Queue, Temporary Topic, or a Temporary Queue.
Timestamp	Currency	Sets or returns the timestamp of the message.
Туре	String	Sets or returns the message type.

4.23.2 StreamMessage Methods

The **StreamMessage** object includes the following methods:

- Acknowledge on page 99
- ClearBody on page 99
- ClearProperties on page 99
- GetProperty on page 100
- PropertyExists on page 100
- **ReadBoolean** on page 101
- ReadByte on page 101
- ReadBytes on page 101
- ReadChar on page 102
- **ReadDouble Method** on page 102
- ReadFloat on page 102
- **ReadInt** on page 103

- **ReadString** on page 104
- Reset on page 104
- SetProperty on page 105
- WriteBoolean on page 105
- WriteByte on page 105
- WriteBytes on page 106
- WriteChar on page 106
- WriteDouble on page 107
- WriteFloat on page 107
- WriteInt on page 107
- WriteLong on page 108
- WriteObject on page 108

- ReadLong Method on page 103
- ReadObject on page 103
- ReadShort on page 104

- WriteShort on page 108
- WriteString on page 109

Acknowledge

Syntax

Sub Acknowledge()

Description

Acknowledges the receipt of current and previous messages.

Parameters

None.

Return Value

A long value representing the result status. A value of 0 (zero) indicates the function was successful; any other value indicates there was an error.

ClearBody

Syntax

Sub ClearBody()

Description

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

Parameters

None.

Return Value

A long value representing the result status. A value of 0 (zero) indicates the function was successful; any other value indicates there was an error.

ClearProperties

Syntax

```
Sub ClearProperties()
```

Description

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

Parameters

None.

A long value representing the result status. A value of 0 (zero) indicates the function was successful; any other value indicates there was an error.

GetProperty

Syntax

Function GetProperty(name As String)

Description

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

Parameters

Name	Туре	Description
name	String	The name of the property.

Return Value

The value of the specified property.

This method also returns a long value representing the result status. A value of 0 (zero) indicates the function was successful; any other value indicates there was an error.

PropertyExists

Syntax

Function PropertyExists (name As String) As Boolean

Description

Checks whether a value for a specific property exists.

Parameters

Name	Туре	Description
name	String	The name of the property to check.

Return Value

A Boolean indicator of whether a value exists for the specified property. If the property value exists, the return value is **true**; otherwise, it is **false**.

This method also returns a long value representing the result status. A value of 0 (zero) indicates the function was successful; any other value indicates there was an error.

ReadBoolean

Syntax

Function ReadBoolean() As Boolean

Description

Reads a Boolean value from the message stream.

Parameters

None.

Return Value

The value of the Boolean property.

This method also returns a long value representing the result status. A value of 0 (zero) indicates the function was successful; any other value indicates there was an error.

ReadByte

Syntax

Function ReadByte() As Byte

Description

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

Parameters

None.

Return Value

The value of the Byte property.

This method also returns a long value representing the result status. A value of 0 (zero) indicates the function was successful; any other value indicates there was an error.

ReadBytes

Syntax

Function ReadBytes(value, [length As Long]) As Long

Description

Reads a portion of the message stream.

Parameters

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

The value of the bytes read into the buffer.

This method also returns a long value representing the result status. A value of 0 (zero) indicates the function was successful; any other value indicates there was an error.

ReadChar

Syntax

Function ReadChar() As Short

Description

Reads a Unicode character value from the message stream.

Parameter

None.

Return Value

The value of the character property.

This method also returns a long value representing the result status. A value of 0 (zero) indicates the function was successful; any other value indicates there was an error.

ReadDouble Method

Syntax

```
Function ReadDouble() As Double
```

Description

Reads a double value from the message stream.

Parameters

None.

Return Value

The value of the double property.

This method also returns a long value representing the result status. A value of 0 (zero) indicates the function was successful; any other value indicates there was an error.

ReadFloat

Syntax

```
Function ReadFloat() As Single
```

Description

Reads a floating point value from the message stream.

Parameters

None.

Return Value

The value of the floating point property.

This method also returns a long value representing the result status. A value of 0 (zero) indicates the function was successful; any other value indicates there was an error.

ReadInt

Syntax

```
Function ReadInt() As Long
```

Description

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

Parameters

None.

Return Value

The value of the integer property.

This method also returns a long value representing the result status. A value of 0 (zero) indicates the function was successful; any other value indicates there was an error.

ReadLong Method

Syntax

```
Function ReadLong() As Currency
```

Description

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

Parameters

None.

Return Value

The value of the long property.

This method also returns a long value representing the result status. A value of 0 (zero) indicates the function was successful; any other value indicates there was an error.

ReadObject

This method is not currently supported.

ReadShort

Syntax

Function ReadShort() As Integer

Description

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

Parameters

None.

Return Value

The value of the short property.

This method also returns a long value representing the result status. A value of 0 (zero) indicates the function was successful; any other value indicates there was an error.

ReadString

Syntax

```
Function ReadString() As String
```

Description

Reads in a string from the stream message.

Parameters

None.

Return Value

The value of the String property.

This method also returns a long value representing the result status. A value of 0 (zero) indicates the function was successful; any other value indicates there was an error.

Reset

Syntax

```
Sub Reset()
```

Description

Places the message body in read-only mode and repositions the stream to the beginning.

Parameters

None.

A long value representing the result status. A value of 0 (zero) indicates the function was successful; any other value indicates there was an error.

SetProperty

Syntax

Sub SetProperty(name As String, value)

Description

Sets a Visual Basic data type value into the message property with the specified name.

Parameters

Name	Туре	Description
name	String	The name of the property.
value	Variant	The value to set in the specified property.

Return Value

A long value representing the result status. A value of 0 (zero) indicates the function was successful; any other value indicates there was an error.

WriteBoolean

Syntax

Sub WriteBoolean(value As Boolean)

Description

Writes a Boolean value to the message stream as a 1-byte value.

Parameters

Name	Туре	Description
value	Boolean	The value to write.

Return Value

A long value representing the result status. A value of 0 (zero) indicates the function was successful; any other value indicates there was an error.

WriteByte

Syntax

Sub WriteByte(value As Byte)

Description

Writes a 1-byte value to the message stream.

Parameters

Name	Туре	Description
value	Byte	The value to write.

Return Value

A long value representing the result status. A value of 0 (zero) indicates the function was successful; any other value indicates there was an error.

WriteBytes

Syntax

Sub WriteBytes(value, [offset], [length])

Description

Writes a byte array, or a portion of the byte array, to the message stream.

Parameters

Name	Туре	Description
value	Variant	The byte array value to write.
offset	Variant	The initial offset within the byte array.
length	Variant	The number of bytes to use.

Return Value

A long value representing the result status. A value of 0 (zero) indicates the function was successful; any other value indicates there was an error.

WriteChar

Syntax

Sub WriteChar(value As Short)

Description

Writes a char value to the message stream as a 2-byte value, high byte first.

Parameters

Name	Туре	Description
value	Short	The value to write.

A long value representing the result status. A value of 0 (zero) indicates the function was successful; any other value indicates there was an error.

WriteDouble

Syntax

Sub WriteDouble(value As Double)

Description

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

Parameters

N	lame	Туре	Description
value		Double	The value to write.

Return Value

A long value representing the result status. A value of 0 (zero) indicates the function was successful; any other value indicates there was an error.

WriteFloat

Syntax

Sub WriteFloat(value As Single)

Description

Converts the floating point argument to a long value, and then writes a 4-byte long value to the message stream (high byte is written first).

Parameters

Name	Туре	Description
value	Single	The value to write.

Return Value

A long value representing the result status. A value of 0 (zero) indicates the function was successful; any other value indicates there was an error.

WriteInt

Syntax

Sub WriteInt(value As Long)

Description

Writes a number to the message stream as a 4-byte integer (high byte is written first).

Parameters

Name	Туре	Description
value	Long	The value to write.

Return Value

A long value representing the result status. A value of 0 (zero) indicates the function was successful; any other value indicates there was an error.

WriteLong

Syntax

Sub WriteLong(value As Currency)

Description

Writes a number to the message stream as an 8-byte long value (high byte is written first).

Parameters

Name	Туре	Description
value	Currency	The value to write.

Return Value

A long value representing the result status. A value of 0 (zero) indicates the function was successful; any other value indicates there was an error.

WriteObject

This method is not currently supported.

WriteShort

Syntax

Sub WriteShort(value As Integer)

Description

Writes a number to the message stream as a 2-byte short integer (high byte is written first).

Parameters

Name	Туре	Description
value	Integer	The value to write.

Return Value

A long value representing the result status. A value of 0 (zero) indicates the function was successful; any other value indicates there was an error.

WriteString

Syntax

Sub WriteString(value As String)

Description

Writes a string to the message stream.

Parameters

Name	Туре	Description
value	String	The value to write.

Return Value

A long value representing the result status. A value of 0 (zero) indicates the function was successful; any other value indicates there was an error.

4.24 TemporaryQueue Object

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

4.24.1 Temporary Queue Properties

The **TemporaryQueue** object includes the property listed in Table 22.

Table 22 Properties of the TemporaryQueue Object

Property	Data Type	Description
QueueName	String	Returns the name of the queue. This property is read-only.

4.24.2 Temporary Queue Object Methods

The TemporaryQueue object includes the following methods:

- Delete on page 110
- ToString on page 110

Delete

Syntax

Sub Delete()

Description

Deletes the temporary queue.

Parameters

None.

Return Value

A long value representing the result status. A value of 0 (zero) indicates the function was successful; any other value indicates there was an error.

ToString

Syntax

Function ToString() As String

Description

Returns a printed version of the queue name.

Parameters

None.

Return Value

A String representation of the queue name.

This method also returns a long value representing the result status. A value of 0 (zero) indicates the function was successful; any other value indicates there was an error.

4.25 TemporaryTopic Object

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

4.25.1 TemporaryTopic Properties

The **TemporaryTopic** object includes the property listed in Table 23.

Table 23 Properties of the TemporaryTopic Object

Property	Data Type	Description
TopicName	String	Returns the name of the topic. This property is read-only.

4.25.2 TemporaryTopic Methods

The **TemporaryTopic** object includes the following methods:

- Delete on page 111
- ToString on page 111

Delete

Syntax

Sub Delete()

Description

Deletes the temporary topic.

Parameters

None.

Return Value

A long value representing the result status. A value of 0 (zero) indicates the function was successful; any other value indicates there was an error.

ToString

Syntax

Function ToString() As String

Description

Returns a printed version of the topic name.

Parameters

None.

Return Value

A String representation of the topic name.

This method also returns a long value representing the result status. A value of 0 (zero) indicates the function was successful; any other value indicates there was an error.

4.26 TextMessage Object

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

4.26.1 TextMessage Properties

The **TextMessage** object includes the properties listed in Table 24.

 Table 24
 Properties of the TextMessage Object

Property	Data Type	Description
CorrelationID	String	Sets or returns correlation ID values that are either JMS IQ Manager message IDs or application-specific strings.
CorrelationIDAsBytes	Variant	Sets or returns the correlation ID as an array of bytes for the message.
DeliveryMode	DeliveryModeConstants	Sets or returns the delivery mode for this message as either msNonPersistent, or msPersistent. The default value is msDefaultDeliveryMode (msPersistent). See DeliveryMode Constants on page 43.
Destination	Destination	Sets or returns the destination for this message.
Expiration	Currency	Sets or returns the message expiration time in milliseconds.
MessageID	String	Sets or returns the value of the uniquely assigned identifier in the message header.
Priority	PriorityConstants	Sets or returns the priority assigned to the message. Possible values are 1 - 9.
Redelivered	Boolean	Sets or returns an indicator of whether the message is redelivered. A value of true indicates the message is redelivered; a value of false indicates it is not.
ReplyTo	Destination	Sets or returns the destination to which a reply to the message is sent. A destination can be a Topic, Queue, Temporary Topic, or a Temporary Queue.
Text	String	Sets or returns the text of the message.
Timestamp	Currency	Sets or returns the timestamp of the message.
Туре	String	Sets or returns the message type.

4.26.2 TextMessage Methods

The **TextMessage** object includes the following methods:

- Acknowledge on page 113
- ClearBody on page 113
- ClearProperties on page 113

- GetProperty on page 114
- PropertyExists on page 114
- SetProperty on page 115

Acknowledge

Syntax

```
Sub Acknowledge()
```

Description

Acknowledges the receipt of current and previous messages.

Parameters

None.

Return Value

A long value representing the result status. A value of 0 (zero) indicates the function was successful; any other value indicates there was an error.

ClearBody

Syntax

```
Sub ClearBody()
```

Description

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

Parameters

None.

Return Value

A long value representing the result status. A value of 0 (zero) indicates the function was successful; any other value indicates there was an error.

ClearProperties

Syntax

Sub ClearProperties()

Description

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

Parameters

None.

Return Value

A long value representing the result status. A value of 0 (zero) indicates the function was successful; any other value indicates there was an error.

GetProperty

Syntax

Function GetProperty(name As String)

Description

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

Parameters

Name	Туре	Description
name	String	The name of the property.

Return Value

The value of the specified property.

This method also returns a long value representing the result status. A value of 0 (zero) indicates the function was successful; any other value indicates there was an error.

PropertyExists

Syntax

Function PropertyExists(name As String) As Boolean

Description

Checks whether a value for a specific property exists.

Parameters

Name	Туре	Description
name	String	The name of the property to check.

Return Value

A Boolean indicator of whether a value exists for the specified property. If the property value exists, the return value is **true**; otherwise, it is **false**.

This method also returns a long value representing the result status. A value of 0 (zero) indicates the function was successful; any other value indicates there was an error.

SetProperty

Syntax

Sub SetProperty(name As String, value)

Description

Sets a Visual Basic data type value into the message property with the specified name.

Parameters

Name	Туре	Description
name	String	The name of the property.
value	Variant	The value to set in the specified property.

Return Value

A long value representing the result status. A value of 0 (zero) indicates the function was successful; any other value indicates there was an error.

4.27 Topic Object

A **Topic** object encapsulates a message service-specific topic name.

4.27.1 Topic Properties

The **Topic** object includes the property listed in Table 25.

 Table 25
 Properties of the Topic Object

Property	Data Type	Description
TopicName	String	Returns the name of the topic. This property is read-only.

4.27.2 Topic Methods

The **Topic** object includes the following method:

ToString on page 116

ToString

Syntax

Function ToString() As String

Description

Returns a printed version of the topic name.

Parameters

None.

Return Value

A String representation of the topic name.

This method also returns a long value representing the result status. A value of 0 (zero) indicates the function was successful; any other value indicates there was an error.

4.28 TopicConnection Object

A **TopicConnection** is an active connection to a pub/sub message service.

4.28.1 TopicConnection Properties

The **TopicConnection** object includes the properties listed in Table 26.

 Table 26
 Properties of the TopicConnection Object

Property	Data Type	Description
ClientID	String	Sets or returns a client identifier for the connection.
MetaData	ConnectionMetaData	This is not currently supported.

4.28.2 TopicConnection Methods

The **TopicConnection** object includes the following methods:

- CreateTopicSession on page 116
- Start on page 117
- Stop on page 117

CreateTopicSession

Syntax

Function CreateTopicSession(Transacted As Boolean, acknowledgeMode As AcknowledgeModeConstants) As TopicSession

Description

Creates a **TopicSession** using the specified transacted and acknowledge modes.

Parameters

Name	Туре	Description
Transacted	Boolean	An indicator of the whether the session is transacted. A value of true indicates the session is transacted; false means it is not.
acknowledgeMode	AcknowledgeModeConstants	An indicator of the acknowledgement mode. See AcknowledgeMode Constants on page 42 for possible values.

Return Value

A topic session.

This method also returns a long value representing the result status. A value of 0 (zero) indicates the function was successful; any other value indicates there was an error.

Start

Syntax

Sub Start()

Description

Starts or restarts a connection's delivery of incoming messages.

Parameters

None.

Return Value

A long value representing the result status. A value of 0 (zero) indicates the function was successful; any other value indicates there was an error.

Stop

Syntax

Function Stop()

Description

Temporarily stops a connection's delivery of incoming messages.

Parameters

None.

Return Value

A long value representing the result status. A value of 0 (zero) indicates the function was successful; any other value indicates there was an error.

4.29 TopicConnectionFactory Object

A client uses a **TopicConnectionFactory** to create a **TopicConnection** for a pub/sub message service.

4.29.1 TopicConnectionFactory Properties

The **TopicConnectionFactory** object includes the properties listed in Table 27.

 Table 27
 Properties of the TopicConnectionFactory Object

Property	Data Type	Description
HostName	String	Sets or returns the name of the host where the message service is running.
Port	Long	Sets or returns the port number at which the message service is listening. The default value is 24053 .
PortOffset	Long	Sets or returns the port offset number of the message service if more then one service is running on the same host machine and using the same port number.

4.29.2 TopicConnectionFactory Methods

The **TopicConnectionFactory** object includes the following method:

- CreateTopicConnection on page 118
- CreateTopicConnectionEx on page 119

CreateTopicConnection

Syntax

Function CreateTopicConnection() As TopicConnection

Description

Creates a topic connection with a default user identity.

Parameters

None.

Return Value

A topic connection.

This method also returns a long value representing the result status. A value of 0 (zero) indicates the function was successful; any other value indicates there was an error.

CreateTopicConnectionEx

Syntax

Function CreateTopicConnection(userId As String, password As String) As TopicConnection

Description

Creates a topic connection with the specified user identity.

Parameters

Name	Туре	Description
userId	String	The logon user ID.
password	String	The password associated with the specified user ID.

Return Value

A topic connection.

This method also returns a long value representing the result status. A value of 0 (zero) indicates the function was successful; any other value indicates there was an error.

4.30 TopicPublisher Object

A client uses a **TopicPublisher** for publishing messages on a topic.

4.30.1 TopicPublisher Properties

The **TopicPublisher** object includes the properties listed in Table 28.

Table 28 Properties of the TopicPublisher Object

Property	Data Type	Description
DeliveryMode	DeliveryModeConstant s	Sets or returns the delivery mode for this message as either msNonPersistent, or msPersistent. The default value is msDefaultDeliveryMode (msPersistent). See DeliveryMode Constants on page 43.

 Table 28
 Properties of the TopicPublisher Object

Property	Data Type	Description
DisableMessageID	Boolean	Sets or returns an indicator of whether message IDs are disabled. A value of true indicates message IDs are disabled; false indicates they are not.
DisableMessageTimestamp	Boolean	Sets or returns an indicator of whether timestamping is disabled. A value of true indicates timestamping is disabled; false indicates it is not.
Priority	PriorityConstants	This is not currently supported.
TimeToLive	Currency	Sets or returns 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 (0 (zero), which is unlimited).
Topic	Topic	Returns the topic associated with the topic publisher.

4.30.2 Topic Publisher Methods

The **TopicPublisher** object includes the following method:

• Publish on page 120

Publish

Syntax

Sub Publish(message As message, [DeliveryMode], [Priority], [TimeToLive], [Topic])

Description

Publishes a message to a topic for an unidentified message producer, specifying delivery mode, priority, time to live, and receiving topic.

Parameters

Name	Туре	Description
message	message	The message to publish.
DeliveryMode	Variant	The delivery mode for the message. See DeliveryMode Constants on page 43.
Priority	Variant	The priority for the message (not currently supported).
TimeToLive	Variant	The lifetime of the message in milliseconds.

Name	Туре	Description
Topic	Variant	The topic to which the message should be published.

Return Value

A long value representing the result status. A value of 0 (zero) indicates the function was successful; any other value indicates there was an error.

4.31 TopicRequestor Object

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

4.31.1 TopicRequestor Methods

The **TopicRequestor** object includes the following methods:

- Close on page 121
- Create on page 121
- Request on page 122

Close

Syntax

Sub Close()

Description

Closes the topic requestor.

Parameters

None.

Return Value

A long value representing the result status. A value of 0 (zero) indicates the function was successful; any other value indicates there was an error.

Create

Syntax

Sub Create(session As TopicSession, Topic As Topic)

Description

Creates a topic requestor.

Parameters

Name	Туре	Description
session	TopicSession	The topic session.
Topic	Topic	The topic associated with the topic requestor.

Return Value

A long value representing the result status. A value of 0 (zero) indicates the function was successful; any other value indicates there was an error.

Request

Syntax

Function Request(message As message, [timeOut]) As message

Description

Sends a request to the topic and waits for a reply, optionally for a specified amount of time.

Parameters

Name	Туре	Description
message	message	The message to send.
timeOut	Variant	The timeout value for the message in milliseconds.

Return Value

A reply message.

This method also returns a long value representing the result status. A value of 0 (zero) indicates the function was successful; any other value indicates there was an error.

4.32 TopicSession Object

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

4.32.1 TopicSession Properties

The **TopicSession** object includes the properties listed in Table 29.

 Table 29
 Properties of the TopicSession Object

Property	Data Type	Description
MessageListener	MessageListener	This is not currently supported.
Transacted	Boolean	Returns an indicator of whether the session is in transacted mode. A value of true indicates the session is in transacted mode; a value of false indicates it is not. This property is readonly.

4.32.2 TopicSession Methods

The **TopicSession** object includes the following methods:

- Commit on page 123
- CreateBytesMessage on page 124
- CreateDurableSubscriber on page 124
- CreateMapMessage on page 125
- CreateMessage on page 125
- CreatePublisher on page 125
- CreateStreamMessage on page 126
- CreateSubscriber on page 126

- CreateTemporaryTopic on page 127
- CreateTextMessage on page 127
- CreateTopic on page 127
- Recover on page 128
- Rollback on page 128
- Run on page 129
- Unsubscribe on page 129

Commit

Syntax

Sub Commit()

Description

Commits all messages processed in this transaction and releases any resources currently held.

Parameters

None.

Return Value

A long value representing the result status. A value of 0 (zero) indicates the function was successful; any other value indicates there was an error.

CreateBytesMessage

Syntax

Function CreateBytesMessage() As BytesMessage

Description

Creates a BytesMessage object (see "BytesMessage Object" on page 44).

Parameters

None.

Return Value

A BytesMessage object.

This method also returns a long value representing the result status. A value of 0 (zero) indicates the function was successful; any other value indicates there was an error.

CreateDurableSubscriber

Syntax

Function CreateDurableSubscriber(Topic As Topic, name As String, [MessageSelector], [NoLocal]) As TopicSubscriber

Description

Creates a durable subscriber for the specified topic.

Parameters

Name	Туре	Description
Topic	Topic	The non-temporary topic to which to subscribe.
name	String	The name used to identify this subscription.
MessageSelector	Variant	The message selector expression that a message's properties must match in order to be delivered. Null can be used.
NoLocal	Variant	An indicator of whether messages published by the topic subscriber's own connection can be delivered. Set this value to true to prevent the delivery of messages published by the topics own connection; set it to false to allow delivery.

Return Value

A **TopicSubscriber** object.

This method also returns a long value representing the result status. A value of 0 (zero) indicates the function was successful; any other value indicates there was an error.

CreateMapMessage

Syntax

Function CreateMapMessage() As MapMessage

Description

Creates a MapMessage object (see "MapMessage Object" on page 59).

Parameters

None.

Return Value

A **MapMessage** object.

This method also returns a long value representing the result status. A value of 0 (zero) indicates the function was successful; any other value indicates there was an error.

CreateMessage

Syntax

Function CreateMessage() As message

Description

Creates a message object (see "message Object" on page 72).

Parameters

None.

Return Value

A **message** object.

This method also returns a long value representing the result status. A value of 0 (zero) indicates the function was successful; any other value indicates there was an error.

CreatePublisher

Syntax

Function CreatePublisher(Topic As Topic) As TopicPublisher

Description

Creates a **TopicPublisher** to publish messages to the specified topic.

Parameters

Name	Туре	Description
Topic	Topic	The topic to which to publish, or null, if this is an unidentified producer.

Return Value

A **TopicPublisher** object.

This method also returns a long value representing the result status. A value of 0 (zero) indicates the function was successful; any other value indicates there was an error.

CreateStreamMessage

Syntax

Function CreateStreamMessage() As StreamMessage

Description

Creates a **StreamMessage** Object" on page 97).

Parameters

None.

Return Value

A StreamMessage object.

This method also returns a long value representing the result status. A value of 0 (zero) indicates the function was successful; any other value indicates there was an error.

CreateSubscriber

Syntax

Function CreateSubscriber(Topic As Topic, [MessageSelector], [NoLocal]) As TopicSubscriber

Description

Creates a non-durable subscriber to the specified topic.

Parameters

Name	Туре	Description
Topic	Topic	The topic to which to subscribe.
MessageSelector	Variant	The message selector expression that a message's properties must match in order to be delivered. Null can be used.
NoLocal	Variant	An indicator of whether messages published by the topic subscriber's own connection can be delivered. Set this value to true to prevent the delivery of messages published by the topics own connection; set it to false to allow delivery.

Return Value

A **TopicSubscriber** object.

This method also returns a long value representing the result status. A value of 0 (zero) indicates the function was successful; any other value indicates there was an error.

CreateTemporaryTopic

Syntax

Function CreateTemporaryTopic() As TemporaryTopic

Description

Creates a temporary topic.

Parameters

None.

Return Value

A temporary topic.

This method also returns a long value representing the result status. A value of 0 (zero) indicates the function was successful; any other value indicates there was an error.

CreateTextMessage

Syntax

Function CreateTextMessage([Text]) As TextMessage

Description

Creates a **TextMessage** object (see "**TextMessage Object**" on page 112).

Parameters

Name	Туре	Description
Text	Variant	The string used to initialize the message.

Return Value

A **TextMessage** object.

This method also returns a long value representing the result status. A value of 0 (zero) indicates the function was successful; any other value indicates there was an error.

CreateTopic

Syntax

Function CreateTopic(TopicName As String) As Topic

Description

Creates a topic identity given a topic name.

Parameters

Name	Туре	Description
TopicName	String	The name of the topic to create.

Return Value

A **Topic** object.

This method also returns a long value representing the result status. A value of 0 (zero) indicates the function was successful; any other value indicates there was an error.

Recover

Syntax

Sub Recover()

Description

Stops message delivery in this session, and then restarts message delivery with the oldest unacknowledged message.

Parameters

None.

Return Value

A long value representing the result status. A value of 0 (zero) indicates the function was successful; any other value indicates there was an error.

Rollback

Syntax

Sub Rollback()

Description

Rolls back any messages processed in this transaction and releases any locks currently held.

Parameters

None.

Return Value

A long value representing the result status. A value of 0 (zero) indicates the function was successful; any other value indicates there was an error.

Run

Syntax

Sub Run()

Description

An optional operation that is only intended to be used by Application Servers and not by ordinary JMS clients.

Parameters

None.

Return Value

A long value representing the result status. A value of 0 (zero) indicates the function was successful; any other value indicates there was an error.

Unsubscribe

Syntax

Sub Unsubscribe (name As String)

Description

Unsubscribes a durable subscription that was created by a client.

Parameters

Name	Туре	Description
name	String	The name used to identify this subscription.

Return Value

A long value representing the result status. A value of 0 (zero) indicates the function was successful; any other value indicates there was an error.

4.33 TopicSubscriber Object

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

4.33.1 TopicSubscriber Properties

The **TopicSubscriber** object includes the properties listed in Table 30.

 Table 30
 Properties of the TopicSubscriber Object

Property	Data Type	Description
MessageListener	MessageListener	This is not currently supported.
MessageSelector	String	Returns the message selector expression for the message consumer. This property is read-only.
NoLocal	Boolean	Returns an indicator of whether messages published by the topic subscriber's own connection can be delivered. A value of true to inhibits the delivery of messages published by its own connection; false allows delivery of such messages. This property is readonly.
Topic	Topic	Returns the topic associated with the topic subscriber.

4.33.2 TopicSubscriber Methods

The **TopicSubscriber** object includes the following methods:

- Close on page 130
- Receive on page 131
- ReceiveNoWait on page 131

Close

Syntax

Sub Close()

Description

Closes the topic subscriber. A message service might allocate resources on behalf of a message consumer, so the receiver should be closed when it is no longer needed.

Parameters

None.

Return Value

A long value representing the result status. A value of 0 (zero) indicates the function was successful; any other value indicates there was an error.

Receive

Syntax

Function Receive([timeOut]) As message

Description

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

Parameters

Name	Туре	Description
timeOut	Variant	The timeout value for the message consumer in milliseconds.

Return Value

A message object.

This method also returns a long value representing the result status. A value of 0 (zero) indicates the function was successful; any other value indicates there was an error.

ReceiveNoWait

Syntax

Function ReceiveNoWait() As message

Description

Receives the next message if one is immediately available.

Parameters

None

Return Value

A **message** object.

This method also returns a long value representing the result status. A value of 0 (zero) indicates the function was successful; any other value indicates there was an error.

4.34 XAQueueConnection Object

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

4.34.1 XAQueueConnection Properties

The **XAQueueConnection** object includes the properties listed in Table 31.

 Table 31
 Properties of the XAQueueConnection Object

Property	Data Type	Description
ClientID	String	Sets or returns the client identifier for the connection.
Metadata	ConnectionMetaData	This is not currently supported.

4.34.2 XAQueueConnection Methods

The **XAQueueConnection** object includes the following methods:

- CreateQueueSession on page 132
- CreateXAQueueSession on page 133
- Start on page 133
- Stop on page 133

CreateQueueSession

Syntax

Function CreateQueueSession($Transacted\ As\ Boolean$, $acknowledgeMode\ As\ AcknowledgeModeConstants$) As QueueSession

Description

Creates a **QueueSession** using the specified transacted and acknowledge modes.

Parameters

Name	Туре	Description
Transacted	Boolean	An indicator of the whether the session is transacted. A value of true indicates the session is transacted; false means it is not.
acknowledgeMode	AcknowledgeModeConstants	An indicator of the acknowledgement mode. See AcknowledgeMode Constants on page 42 for possible values.

Return Value

A queue session.

This method also returns a long value representing the result status. A value of 0 (zero) indicates the function was successful; any other value indicates there was an error.

CreateXAQueueSession

Syntax

Function CreateXAQueueSession() As XAQueueSession

Description

Creates an **XAQueueSession** object.

Parameters

None.

Return Value

A long value representing the result status. A value of 0 (zero) indicates the function was successful; any other value indicates there was an error.

Start

Syntax

```
Sub Start()
```

Description

Starts or restarts a connection's delivery of incoming messages.

Parameters

None.

Return Value

A long value representing the result status. A value of 0 (zero) indicates the function was successful; any other value indicates there was an error.

Stop

Syntax

```
Sub Stop()
```

Description

Temporarily stops a connection's delivery of incoming messages.

Parameters

None.

Return Value

A long value representing the result status. A value of 0 (zero) indicates the function was successful; any other value indicates there was an error.

4.35 XAQueueConnectionFactory Object

An **XAQueueConnectionFactory** object provides the same create options as **QueueConnectionFactory**. The only difference is that an **XAQueueConnectionFactory** object is transacted by definition.

4.35.1 XAQueueConnectionFactory Object Properties

The **XAQueueConnectionFactory** object includes the properties listed in Table 32.

Table 32	Properties of the XAC	QueueConnectionFactor	y Object
----------	-----------------------	-----------------------	----------

Property	Data Type	Description
HostName	String	Sets or returns the name of the host where the message service is running.
Port	Long	Sets or returns the port number at which the message service is listening. The default value is 24053 .
PortOffset	Long	Sets or returns the port offset number of the message service if more then one service is running on the same host machine and using the same port number.

4.35.2 XAQueueConnectionFactory Methods

The **XAQueueConnectionFactory** object includes the following methods:

- CreateQueueConnection on page 134
- CreateQueueConnectionEx on page 135
- CreateXAQueueConnection on page 135
- CreateXAQueueConnectionEx on page 135

CreateQueueConnection

Syntax

Function CreateQueueConnection() As QueueConnection

Description

Creates a queue connection with a default user identity.

Parameters

None.

Return Value

A queue connection.

This method also returns a long value representing the result status. A value of 0 (zero) indicates the function was successful; any other value indicates there was an error.

CreateQueueConnectionEx

Syntax

Function CreateQueueConnection($userId\ As\ String$, $password\ As\ String$) As QueueConnection

Description

Creates a queue connection with the specified user identity.

Parameters

Name	Туре	Description
userId	String	The logon user ID.
password	String	The password associated with the specified user ID.

Return Value

A queue connection.

This method also returns a long value representing the result status. A value of 0 (zero) indicates the function was successful; any other value indicates there was an error.

CreateXAQueueConnection

Syntax

Function CreateXAQueueConnection() As XAQueueConnection

Description

Creates an XA queue connection with a default user identity.

Parameters

None.

Return Value

An XA queue connection.

This method also returns a long value representing the result status. A value of 0 (zero) indicates the function was successful; any other value indicates there was an error.

CreateXAQueueConnectionEx

Syntax

Function CreateQueueConnection($userId\ As\ String$, $password\ As\ String$) As XAQueueConnection

Description

Creates an XA queue connection with the specified user identity.

Parameters

Name	Туре	Description
userId	String	The logon user ID.
password	String	The password associated with the specified user ID.

Return Value

An XA queue connection.

This method also returns a long value representing the result status. A value of 0 (zero) indicates the function was successful; any other value indicates there was an error.

4.36 XAQueueSession Object

An **XAQueueSession** object provides a regular **QueueSession**, with the exception that it is transacted by definition. It can be used to create the following objects: **QueueReceiver**, **QueueSender**, and **QueueBrowser**.

4.36.1 XAQueueSession Properties

The **XAQueueSession** object includes the properties listed in Table 33.

Table 33 Properties of the XAQueueSession Object

Property	Data Type	Description
MessageListener	MessageListener	This is not currently supported.
QueueSession	QueueSession	Returns the queue session associated with the XAQueueSession . This property is read-only.
Transacted	Boolean	Returns an indicator of whether the session is in transacted mode. A value of true indicates the session is in transacted mode; a value of false indicates it is not. This property is readonly.

4.36.2 XAQueueSession Methods

The **XAQueueSession** object includes the following methods:

Commit on page 137

CreateTextMessage on page 138

- CreateBytesMessage on page 137
- CreateMapMessage on page 137
- CreateMessage on page 138
- CreateStreamMessage on page 138
- Recover on page 139
- Rollback on page 139
- Run on page 140

Commit

Syntax

Sub Commit()

Description

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

Parameters

None.

Return Value

A long value representing the result status. A value of 0 (zero) indicates the function was successful; any other value indicates there was an error.

CreateBytesMessage

Syntax

Function CreateBytesMessage() As BytesMessage

Description

Creates a BytesMessage object (see "BytesMessage Object" on page 44).

Parameters

None.

Return Value

A **BytesMessage** object.

This method also returns a long value representing the result status. A value of 0 (zero) indicates the function was successful; any other value indicates there was an error.

CreateMapMessage

Syntax

Function CreateMapMessage() As MapMessage

Description

Creates a MapMessage object (see "MapMessage Object" on page 59).

Parameters

None.

Return Value

A **MapMessage** object.

This method also returns a long value representing the result status. A value of 0 (zero) indicates the function was successful; any other value indicates there was an error.

CreateMessage

Syntax

```
Function CreateMessage() As message
```

Description

Creates a message object (see "message Object" on page 72).

Parameters

None.

Return Value

A message object.

This method also returns a long value representing the result status. A value of 0 (zero) indicates the function was successful; any other value indicates there was an error.

CreateStreamMessage

Syntax

```
Function CreateStreamMessage() As StreamMessage
```

Description

Creates a **StreamMessage** object (see "**StreamMessage Object**" on page 97).

Parameters

None.

Return Value

A StreamMessage object.

This method also returns a long value representing the result status. A value of 0 (zero) indicates the function was successful; any other value indicates there was an error.

CreateTextMessage

Syntax

Function CreateTextMessage([Text]) As TextMessage

Description

Creates a TextMessage object (see "TextMessage Object" on page 112).

Parameters

Name	Туре	Description
Text	Variant	The string used to initialize the message.

Return Value

A **TextMessage** object.

This method also returns a long value representing the result status. A value of 0 (zero) indicates the function was successful; any other value indicates there was an error.

Recover

Syntax

Sub Recover()

Description

Stops message delivery in this session, and then restarts message delivery with the oldest unacknowledged message.

Parameters

None.

Return Value

A long value representing the result status. A value of 0 (zero) indicates the function was successful; any other value indicates there was an error.

Rollback

Syntax

Sub Rollback()

Description

Rolls back any messages processed in this transaction and releases any locks currently held.

Parameters

None.

Return Value

A long value representing the result status. A value of 0 (zero) indicates the function was successful; any other value indicates there was an error.

Run

Syntax

Sub Run()

Description

An optional operation that is only intended to be used by Application Servers and not by ordinary JMS clients.

Parameters

None.

Return Value

A long value representing the result status. A value of 0 (zero) indicates the function was successful; any other value indicates there was an error.

4.37 XASession Object

The **XASession** object extends the capability of **session** by adding access to a message service's support for transacted messages using the Compensating Resource Manager (CRM), handled under the Distributed Transaction Coordinator (DTC).

4.37.1 XASession Object Properties

The **XASession** object includes the properties listed in Table 34.

Table 34 Properties of the XASession Object

Property	Data Type	Description
MessageListener	MessageListener	This is not currently supported.
Transacted	Boolean	Returns an indicator of whether the session is in transacted mode. A value of true indicates the session is in transacted mode; a value of false indicates it is not. This property is readonly.

4.37.2 XASession Object Methods

The **XASession** object includes the following methods:

- Commit on page 141
- CreateBytesMessage on page 141
- CreateMapMessage on page 141
- CreateMessage on page 142

- CreateTextMessage on page 142
- Recover on page 143
- Rollback on page 143
- Run on page 144

CreateStreamMessage on page 142

Commit

Syntax

```
Sub Commit()
```

Description

Commits all messages processed in this transaction and releases any locks currently held.

Parameters

None.

Return Value

A long value representing the result status. A value of 0 (zero) indicates the function was successful; any other value indicates there was an error.

CreateBytesMessage

Syntax

```
Function CreateBytesMessage() As BytesMessage
```

Description

Creates a BytesMessage object (see "BytesMessage Object" on page 44).

Parameters

None.

Return Value

A **BytesMessage** object.

This method also returns a long value representing the result status. A value of 0 (zero) indicates the function was successful; any other value indicates there was an error.

CreateMapMessage

Syntax

```
Function CreateMapMessage() As MapMessage
```

Description

Creates a MapMessage object (see "MapMessage Object" on page 59).

Parameters

None.

Return Value

A **MapMessage** object.

This method also returns a long value representing the result status. A value of 0 (zero) indicates the function was successful; any other value indicates there was an error.

CreateMessage

Syntax

Function CreateMessage() As message

Description

Creates a message object (see "message Object" on page 72).

Parameters

None.

Return Value

A message object.

This method also returns a long value representing the result status. A value of 0 (zero) indicates the function was successful; any other value indicates there was an error.

CreateStreamMessage

Syntax

Function CreateStreamMessage() As StreamMessage

Description

Creates a StreamMessage Object (see "StreamMessage Object" on page 97).

Parameters

None.

Return Value

A **StreamMessage** object.

This method also returns a long value representing the result status. A value of 0 (zero) indicates the function was successful; any other value indicates there was an error.

CreateTextMessage

Syntax

Function CreateTextMessage([Text]) As TextMessage

Description

Creates a **TextMessage** object (see "**TextMessage Object**" on page 112).

Parameters

Name	Туре	Description
Text	Variant	The string used to initialize the message.

Return Value

A **TextMessage** object.

This method also returns a long value representing the result status. A value of 0 (zero) indicates the function was successful; any other value indicates there was an error.

Recover

Syntax

Sub Recover()

Description

Stops message delivery for the session, and then restarts message delivery starting with the oldest unacknowledged message.

Parameters

None.

Return Value

A long value representing the result status. A value of 0 (zero) indicates the function was successful; any other value indicates there was an error.

Rollback

Syntax

Sub Rollback()

Description

Rolls back any messages processed in this transaction and releases any locks currently held.

Parameters

None.

Return Value

A long value representing the result status. A value of 0 (zero) indicates the function was successful; any other value indicates there was an error.

Run

Syntax

Sub Run()

Description

An optional operation that is only intended to be used by Application Servers and not by ordinary JMS clients.

Parameters

None.

Return Value

A long value representing the result status. A value of 0 (zero) indicates the function was successful; any other value indicates there was an error.

4.38 XATopicConnection Object

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

4.38.1 XATopicConnection Properties

The **XATopicConnection** object includes the properties listed in Table 35.

Table 35 Properties of the XATopicConnection Object

Property	Data Type	Description
ClientID	String	Sets or returns a client identifier for the connection.
MetaData	ConnectionMetaData	This is not currently supported.

4.38.2 XATopicConnection Methods

The **XATopicConnection** object includes the following methods:

- CreateTopicSession on page 145
- Start on page 146
- Stop on page 146

CreateTopicSession

Syntax

Function CreateTopicSession(Transacted As Boolean, acknowledgeMode As AcknowledgeModeConstants) As TopicSession

Description

Creates a **TopicSession** using the specified transacted and acknowledge modes.

Parameters

Name	Туре	Description
Transacted	Boolean	An indicator of the whether the session is transacted. A value of true indicates the session is transacted; false means it is not.
acknowledgeMode	AcknowledgeModeConstants	An indicator of the acknowledgement mode. See AcknowledgeMode Constants on page 42 for possible values.

Return Value

A topic session.

This method also returns a long value representing the result status. A value of 0 (zero) indicates the function was successful; any other value indicates there was an error.

CreateXATopicSession

Syntax

Function CreateXATopicSession() As XATopicSession

Description

Creates an **XATopicSession** using the specified transacted and acknowledge modes.

Parameters

Name	Туре	Description
Transacted	Boolean	An indicator of the whether the session is transacted. A value of true indicates the session is transacted; false means it is not.
acknowledgeMode	AcknowledgeModeConstants	An indicator of the acknowledgement mode. See AcknowledgeMode Constants on page 42 for possible values.

Return Value

An XA topic session.

This method also returns a long value representing the result status. A value of 0 (zero) indicates the function was successful; any other value indicates there was an error.

Start

Syntax

Sub Start()

Description

Starts or restarts a connection's delivery of incoming messages.

Parameters

None.

Return Value

A long value representing the result status. A value of 0 (zero) indicates the function was successful; any other value indicates there was an error.

Stop

Syntax

Function Stop()

Description

Temporarily stops a connection's delivery of incoming messages.

Parameters

None.

Return Value

A long value representing the result status. A value of 0 (zero) indicates the function was successful; any other value indicates there was an error.

4.39 XATopicConnectionFactory Object

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

4.39.1 XATopicConnectionFactory Properties

The **XATopicConnectionFactory** object includes the properties listed in Table 36. **Table 36** Properties of the XATopicConnectionFactory Object

Property	Data Type	Description
HostName	String	Sets or returns the name of the host where the message service is running.
Port	Long	Sets or returns the port number at which the message service is listening. The default value is 24053 .
PortOffset	Long	Sets or returns the port offset number of the message service if more then one service is running on the same host machine and using the same port number.

4.39.2 XATopicConnectionFactory Methods

The **XATopicConnectionFactory** object includes the following method:

- CreateTopicConnection on page 147
- CreateTopicConnectionEx on page 148
- CreateXATopicConnection on page 148
- CreateXATopicConnectionEx on page 148

CreateTopicConnection

Syntax

Function CreateTopicConnection() As TopicConnection

Description

Creates a topic connection with a default user identity.

Parameters

None.

Return Value

A topic connection.

This method also returns a long value representing the result status. A value of 0 (zero) indicates the function was successful; any other value indicates there was an error.

CreateTopicConnectionEx

Syntax

Function CreateTopicConnection($userId\ As\ String$, $password\ As\ String$) As TopicConnection

Description

Creates a topic connection with the specified user identity.

Parameters

Name	Туре	Description
userId	String	The logon user ID.
password	String	The password associated with the specified user ID.

Return Value

A topic connection.

This method also returns a long value representing the result status. A value of 0 (zero) indicates the function was successful; any other value indicates there was an error.

CreateXATopicConnection

Syntax

Function CreateXATopicConnection() As XATopicConnection

Description

Creates an XA topic connection with a default user identity.

Parameters

None.

Return Value

An XA topic connection.

This method also returns a long value representing the result status. A value of 0 (zero) indicates the function was successful; any other value indicates there was an error.

CreateXATopicConnectionEx

Syntax

Function CreateXATopicConnection(userId As String, password As String) As XATopicConnection

Description

Creates an XA topic connection with the specified user identity.

Parameters

Name	Туре	Description
userId	String	The logon user ID.
password	String	The password associated with the specified user ID.

Return Value

An XA topic connection.

This method also returns a long value representing the result status. A value of 0 (zero) indicates the function was successful; any other value indicates there was an error.

4.40 XATopicSession Object

An **XATopicSession** provides a regular **TopicSession** object, with the exception that it is transacted by definition. It can be used to create **TopicSubscriber** and **TopicPublisher** objects.

4.40.1 XATopicSession Properties

The **XATopicSession** object includes the properties listed in Table 37.

 Table 37
 Properties of the XATopicSession Object

Property	Data Type	Description
MessageListener	MessageListener	This is not currently supported.
TopicSession	TopicSession	Returns the topic session associated with the XATopicSession . This property is read-only.
Transacted	Boolean	Returns an indicator of whether the session is in transacted mode. A value of true indicates the session is in transacted mode; a value of false indicates it is not. This property is readonly.

4.40.2 XATopicSession Methods

The **XATopicSession** object includes the following methods:

- Commit on page 150
- CreateBytesMessage on page 150
- CreateMapMessage on page 150
- CreateTextMessage on page 151
- Recover on page 152
- Rollback on page 152

CreateMessage on page 151

- Run on page 153
- CreateStreamMessage on page 151

Commit

Syntax

Sub Commit()

Description

Commits all messages processed in this transaction and releases any resources currently held.

Parameters

None.

Return Value

A long value representing the result status. A value of 0 (zero) indicates the function was successful; any other value indicates there was an error.

CreateBytesMessage

Syntax

Function CreateBytesMessage() As BytesMessage

Description

Creates a BytesMessage Object (see "BytesMessage Object" on page 44).

Parameters

None.

Return Value

A **BytesMessage** object.

This method also returns a long value representing the result status. A value of 0 (zero) indicates the function was successful; any other value indicates there was an error.

CreateMapMessage

Syntax

Function CreateMapMessage() As MapMessage

Description

Creates a MapMessage object (see "MapMessage Object" on page 59).

Parameters

None.

Return Value

A **MapMessage** object.

This method also returns a long value representing the result status. A value of 0 (zero) indicates the function was successful; any other value indicates there was an error.

CreateMessage

Syntax

Function CreateMessage() As message

Description

Creates a message object (see "message Object" on page 72).

Parameters

None.

Return Value

A message object.

This method also returns a long value representing the result status. A value of 0 (zero) indicates the function was successful; any other value indicates there was an error.

CreateStreamMessage

Syntax

Function CreateStreamMessage() As StreamMessage

Description

Creates a **StreamMessage** Object (see "**StreamMessage Object**" on page 97).

Parameters

None.

Return Value

A **StreamMessage** object.

This method also returns a long value representing the result status. A value of 0 (zero) indicates the function was successful; any other value indicates there was an error.

CreateTextMessage

Syntax

Function CreateTextMessage([Text]) As TextMessage

Description

Creates a **TextMessage** object (see "**TextMessage Object**" on page 112).

Parameters

Name	Туре	Description
Text	Variant	The string used to initialize the message.

Return Value

A **TextMessage** object.

This method also returns a long value representing the result status. A value of 0 (zero) indicates the function was successful; any other value indicates there was an error.

Recover

Syntax

Sub Recover()

Description

Stops message delivery in this session, and then restarts message delivery with the oldest unacknowledged message.

Parameters

None.

Return Value

A long value representing the result status. A value of 0 (zero) indicates the function was successful; any other value indicates there was an error.

Rollback

Syntax

Sub Rollback()

Description

Rolls back any messages processed in this transaction and releases any locks currently held.

Parameters

None.

Return Value

A long value representing the result status. A value of 0 (zero) indicates the function was successful; any other value indicates there was an error.

Run

Syntax

Sub Run()

Description

An optional operation that is only intended to be used by Application Servers and not by ordinary JMS clients.

Parameters

None.

Return Value

A long value representing the result status. A value of 0 (zero) indicates the function was successful; any other value indicates there was an error.

4.41 Error Codes

Common error codes in the COM+ API are given below.

4.41.1 IErrorInfo Methods

Table 38 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.

4.41.2 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 39 HRESULT Values

Value	Description
E_FAIL	Failure.
E_NOTIMPL	Method is not supported.
S_FALSE	Success. Condition was FALSE.
S_OK	Success. Numerically equivalent to NOERROR.

4.41.3 Error Value Constants

Table 40 lists the possible error values and provides an explanation of what each means.

 Table 40
 Error Value Constants

Error Value Constant	Description
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.
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.

 Table 40
 Error Value Constants

Error Value Constant	Description
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.

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

- About the COM+ Samples on page 156
- Implementing the Java CAPS Projects on page 157
- Building the Sample COM+ Application on page 158
- Running the Sample COM+ Applications on page 164
- Building the CRM Sample Application on page 160

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 41. The table describes what each .**zip** file contains.

Table 41 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 41 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 16. Implementing the sample Projects consists of the following steps.

- Importing the Sample Project on page 157
- Creating the Environment on page 158
- Deploying the Projects on page 158

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 159
- Configuring the Sample Environment on page 159
- Building the Sample Applications on page 159

Table 42 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 42
 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**.
- 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 17). 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 160.

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 42 on page 159 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 160
- Configuring and Building the CRM Sample on page 162
- Creating the CRM Sample Application on page 163

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 22 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.

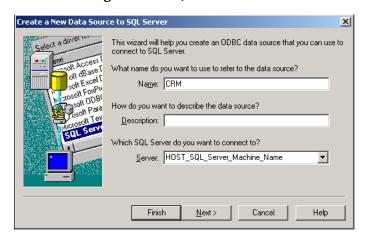
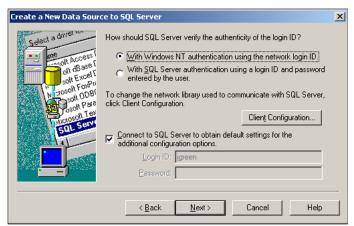


Figure 23 SQL Data source

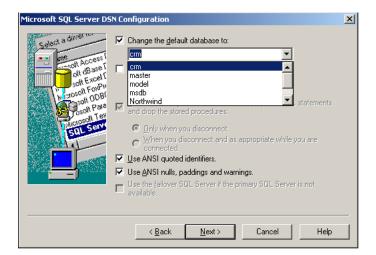
7 Select With Windows authentication using the network logon ID, select Connect to SQL Server to obtain default settings for the additional configuration options, and then click Next.

Figure 24 Login Settings



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

Figure 25 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 COM+ 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 26 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 42 on page 159**.

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.

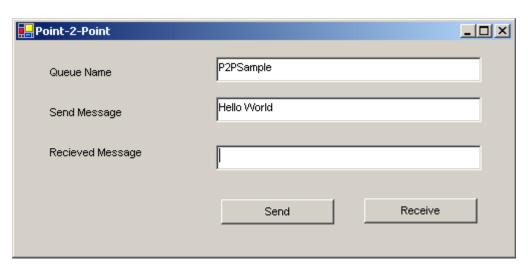


Figure 27 Point 2 Point Sample Dialog

- 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 160.

- 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**.

WriteLong 56 WriteObject 56 WriteShort 56 Index WriteUTF 57 BytesMessage object 44, 72 BytesMessage properties 45 CorrelationID 45 CorrelationIDAsBytes 45 **Symbols** DeliveryMode 45 **Destination 45** .NET 2003 158, 159 **Expiration 45** MessageID 45 Priority 45 Redelivered 45 Acknowledge method 46, 61, 73, 99, 113 ReplyTo 45 acknowledge modes Timestamp 45 AutoAcknowledge 42 Type 45 ClientAcknowledge 43 DupsOKAcknowledge 43 C AcknowledgeMode constants 42 Administrative Tools 160 ClearBody method 46, 61, 74, 99, 113 AutoAcknowledge mode 42 ClearMessage methods ClearProperties 61 B ClearProperties method 47, 61, 74, 99, 113 ClientAcknowledge mode 43 body 21 ClientID property 79, 116, 132, 144 BytesMessage 22 Close method 76, 83, 85, 121, 130 BytesMessage methods 45 CodeSamples.zip 156 Acknowledge 46 CodeSamplesUNIX.tar 156 ClearBody 46 Collaborations 157 ClearProperties 47 COM+ applications 28, 31 GetProperty 47, 66 COM+ library 27 PropertyExists 47 Commit method 88, 94, 123, 137, 141, 150 ReadBoolean 48 compensating resource manager 26 ReadByte 48 about 26 ReadBytes 49 architecture 26 ReadChar 49 configuring 28 ReadDouble 49 implementing 28 ReadFloat 50 two-phase commit protocol 27 ReadInt 50 Component Services 28 ReadLong 50 Connection methods ReadShort 51 Start 58 ReadUnsignedByte 51 Stop 58 ReadUnsignedShort 51 Connection object 57 ReadUTF 52 Connection properties 57 Reset 52 ConnectionFactory object 58 SetProperty 53 ConnectionFactory properties 58 WriteBoolean 53 HostName 58 WriteByte 53, 105 Port 59 WriteBytes 54 PortOffset 59 WriteChar 54 ConnectionMetaData object 59 WriteDouble 55 constants WriteFloat 55 AcknowledgeMode constants 42

WriteInt 55

DeliveryMode constants 43	CRMclient.exe 163
message 43	CRMTest.dll 160, 163
miscellaneous 43	
msDefaultDeliveryMode 44	D
msDefaultPriority 44	D
msDefaultTimeToLive 44	data format 20
Priority 44	defaults
conventions, text 13	Message interface 43
correlation ID 22, 24	Delete method 110, 111
CorrelationID property 45, 59, 72, 97, 112	delivery mode 21
CorrelationIDAsBytes property 45, 59, 72, 97, 112	delivery modes
Create method 85, 121	msNonPersistent 43
CreateBytesMessage method 89, 94, 124, 137, 141,	msPersistent 43
150	DeliveryMode constants 43
CreateDurableSubscriber method 124	DeliveryMode property 45, 59, 72, 78, 86, 97, 112,
CreateMapMessage method 89, 95, 125, 137, 141,	119
150	Deployment Profile 158
CreateMessage method 89, 95, 125, 138, 142, 151	destination 21, 40
CreatePublisher method 125	Destination property 45, 59, 72, 98, 112
CreateQueue method 90	destinations 25, 42
CreateQueueConnection method 82, 134	DisableMessageID property 78, 86, 120
CreateQueueConnectionEx method 82, 135	DisableMessageTimestamp property 78, 86, 120
CreateQueueSession method 80, 132	distributed transaction 26
CreateReceiver method 90	Domain Manager 20
CreateSender method 91	DupsOKAcknowledge mode 43
CreateStreamMessage method 91, 95, 126, 138, 142,	•
151	F.
CreateSubscriber method 126	E
CreateTemporaryQueue method 91	eGateAPIKitDocs.sar, installing 16
CreateTemporaryTopic method 127	Enterprise Designer 156
CreateTextMessage method 92, 96, 127, 139, 142,	Environment 158
151	error codes 153
CreateTopic method 127	expiration 21
CreateTopicConnection method 118, 147	Expiration property 45, 59, 72, 98, 112
CreateTopicConnectionEx method 119, 148	
CreateTopicSession method 116, 145	G
CreateXAQueueConnection method 135	U
CreateXAQueueConnectionEx method 136	GetBoolean method 62
CreateXAQueueSession method 133	GetByte method 62
CreateXATopicConnection method 148	GetBytes method 62
CreateXATopicConnectionEx method 148	GetChar property 63
CreateXATopicSession method 145	GetDouble method 63
CRM 26	GetFloat method 64
about 26 architecture 26	GetInt method 64
	GetLong method 65
configuring 28	GetObject method 65
implementing 28	GetProperty method 47, 65, 74, 100
two-phase commit protocol 27	GetProperty methods 114
CRM sample 160	GetShort method 66
building 160	GetString method 66
configuring 162	
creating the application 163 database 160	
running 166	

H	SetBytes 68
handar fields 21	SetChar 69
header fields 21	SetDouble 69
HostName property 58, 81, 118, 134, 147	SetFloat 69
	SetInt 70
I .	SetLong 70
	SetObject 71
Implementing 157	SetProperty 71
implementing message server models 20	SetShort 71
IQ Manager field 20	SetString 72
IQ Manager SSL field 20	MapMessage object 59, 72
ItemExists method 66	MapMessage properties 59
	CorrelationID 59
T. Control of the Con	CorrelationIDAsBytes 59
J	DeliveryMode 59
Java 20	Destination 59
Java CAPS Projects 157	Expiration 59
JMS API in C++	MapNames 60
differences with Java API 40	MessageID 60
JMS interfaces 18	Priority 60
JMSCorrelationID 22, 24	Redelivered 60
JMSDeliveryMode 21	ReplyTo 60
JMSDestination 21	Timestamp 60
JMSExpiration 21	Type 60
JMSMessageID 21	MapNames property 60
JMSPriority 21	Message constants 44
JMSRedelivered 21	_
JMSReplyTo 22	message consumer 40
JMSTimestamp 21	message consumer 40
,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,	message ID 21 Message interface defaults 43
	Message interface defaults 43
L	message methods 73
Logical Host 19, 158	Acknowledge 73
Logical Host 19, 130	ClearBranarties 74
	ClearProperties 74
M	GetProperty 74
MM	PropertyExists 75
MapMessage methods 60	SetProperty 75
Acknowledge 61	message object 72
ClearBody 61	message producer 40
GetBoolean 62	message properties 72
GetByte 62	CorrelationID 72
GetBytes 62	CorrelationIDAsBytes 72
GetChar 63	DeliveryMode 72
GetDouble 63	Destination 72
GetFloat 64	Expiration 72
GetInt 64	MessageID 73
GetLong 65	Priority 73
GetObject 65	Redelivered 73
GetProperty 65	ReplyTo 73
GetShort 66	Timestamp 73
ItemExists 66	Type 73
PropertyExists 67	message selector 25
SetBoolean 67	message server models 20
SetByte 68	MessageConsumer method

ReceiveNoWait 77	TopicRequestor 121
MessageConsumer methods 76	TopicSession 123
Close 76	TopicSubscriber 130
ReceiveMessage 76	XAQueueConnection 132
MessageConsumer object 76	XAQueueConnectionFactory 134
MessageConsumer properties 76	XAQueueSession 136
MessageListener 76	XASession 140
MessageSelector 76	XATopicConnection 144
MessageID property 45, 60, 73, 98, 112	XATopicConnectionFactory 147
MessageListener object 77	XATopicSession 149
MessageListener property 76, 83, 88, 93, 123, 130,	miscellaneous constants 43
136, 140, 149	msDefaultDeliveryMode constant 44
MessageProducer object 77	msDefaultPriority constant 44
MessageProducer properties 78	msDefaultTimeToLive constant 44
DeliveryMode 78	msNonPersistent mode 43
DisableMessageID 78	msPersistent mode 43
DisableMessageTimestamp 78	multi-threaded apartment model 25
Priority 78	muni uneducu aparament model 20
TimeToLive 78	
messages 40	N
body (payload) 21, 22	NoLocal property 130
header fields 21	NoLocal property 130
JMSCorrelationID 22	
JMSDeliveryMode 21	O
JMSDestination 21	ODDC Data Carray A larga later a 100
JMSExpiration 21	ODBC Data Source Administrator 160
JMSMessageID 21	
JMSPriority 21	P
JMSRedelivered 21	
	payload 21
JMSReplyTo 22	point-to-point messaging 23
JMSTimestamp 21	port number 20
properties 21, 22 Massage Selector property, 76, 82, 130	Port property 59, 81, 118, 134, 147
MessageSelector property 76, 83, 130	PortOffset property 59, 81, 118, 134, 147
MetaData property 79 , 116 , 132 , 144	priority 21
methods Partice Manager 45	Priority constants 44
BytesMessage 45	Priority property 45, 60, 73, 78, 86, 98, 112, 120
MapMessage 60	project samples
message 73	about 156
MessageConsumer 76	building 158 , 159
Queue 79	CRM 160
QueueConnection 80	building 160
QueueConnectionFactory 81	configuring 162
QueueReceiver 83	creating the application 163
QueueRequestor 84	database 160
QueueSender 87	running 166
QueueSession 88	directory structure 159
session 94	environment 159
StreamMessage 98	implementing 157
TemporaryTopic 111	running 164–166
TextMessage 113	XA 160°
Topic 115	properties 21, 22
TopicConnection 116	BytesMessage 45
TopicConnectionFactory 118 TopicPublisher 120	Connection 57

ConnectionFactory 58	HostName 81
MapMessage 59	Port 81
message 72	PortOffset 81
MessageConsumer 76	QueueName property 78, 109
MessageProducer 78	QueueReceiver methods 83
Queue 78	Close 83
QueueConnection 79	Receive 83
QueueConnectionFactory 81	ReceiveNoWait 84, 131
QueueReceiver 83	QueueReceiver object 82
QueueSender 86	QueueReceiver properties 83
QueueSession 88	MessageListener 83
session 93	MessageSelector 83
StreamMessage 97	Queue 83
TemporaryQueue 109	QueueRequestor methods 84
TemporaryTopic 111	Close 85
TextMessage 112	Create 85
Topic 115	Request 85
TopicConnection 116	QueueRequestor object 84
TopicConnectionFactory 118	QueueSender methods 87
TopicPublisher 119	Send 87
TopicSession 123	QueueSender object 86
TopicSubscriber 130	QueueSender properties 86
XAQueueConnection 132	DeliveryMode 86
XAQueueConnectionFactory 134	DisableMessageID 86
XAQueueSession 136	DisableMessageTimestamp 86
XASession 140	Priority 86
XATopicConnectionFactory 147	Queue 86
XATopicSession 144, 149	TimeToLive 87
PropertyExists method 47, 67, 75, 100, 114	QueueSession methods 88
Publish method 120	Commit 88
publish/subscribe messaging 22	CreateBytesMessage 89
	CreateMapMessage 89
0	CreateMessage 89
Q	CreateQueue 90
Queue methods 79	CreateReceiver 90
ToString 79	CreateSender 91
Queue object 78	CreateStreamMessage 91
Queue properties 78	CreateTemporaryQueue 91
QueueName 78	CreateTextMessage 92
Queue property 83, 86	Recover 92
QueueBrowser object 79	Rollback 93
QueueConnection methods 80	Run 93
CreateQueueSession 80	QueueSession object 88
Start 80	QueueSession properties 88
Stop 81	MessageListener 88
QueueConnection object 79	Transacted 88
QueueConnection properties 79	QueueSession property 136
ClientID 79	
MetaData 79	D
QueueConnectionFactory methods 81	R
CreateQueueConnection 82	ReadBoolean method 48, 101
CreateQueueConnectionEx 82	ReadByte method 48, 101
QueueConnectionFactory object 81	ReadBytes method 49, 101
QueueConnectionFactory properties 81	ReadChar method 49, 102

ReadDouble method 49, 102	screenshots 13
ReadFloat method 50, 102	selector 25
ReadInt method 50, 103	Send method 87
ReadLong method 50, 103	send/receive messaging 23
ReadObject method 103	session 40
ReadShort method 51, 104	Session methods
ReadString method 104	CreateMapMessage 125
ReadUnsignedByte method 51	session methods 94
ReadUnsignedShort method 51	Commit 94
ReadUTF method 52	CreateBytesMessage 94
Receive Message method 76	CreateMapMessage 95
Receive method 83, 131	CreateMessage 95
ReceiveNoWait method 77, 84, 131	CreateStreamMessage 95
Recover method 92, 96, 128, 139, 143, 152	CreateTextMessage 96
Redelivered property 45, 60, 73, 98, 112	Recover 96
ReplyTo property 45, 60, 73, 98, 112	Rollback 96
Repository 19	Run 97
Request method 85, 122	session object 93
request/reply messaging 24	session properties 93
requestor 40	MessageListener 93
Reset method 52, 104	Transacted 93
Rollback method 93, 96, 128, 139, 143, 152	SetBoolean method 67
Run method 93, 97, 129, 140, 144, 153	SetByte method 68
	SetBytes method 68
S	SetChar method 69
3	SetDouble method 69
sample code for using JMS	SetFloat methods 69
compensating resource manager (CRM) 160	SetInt method 70
message selector	SetLong method 70
(discussed) 25	SetObject method 71
publish/subscribe	SetProperty method 53, 71, 75, 105, 115
(diagrammed) 23	SetShort method 71
queue send/receive	SetString method 72
(diagrammed) 24	Start method 58, 80, 117, 133, 146
request-reply 24	STC_MSCOM.Compensator 33
(diagrammed) 24	stc_mscom.dll 27, 31, 33, 38
XA 25	STC_MSCOM.XAQueueConnectionFactory 36
sample Projects	STC_MSCOM.XATopicConnectionFactory 36
importing 157	SteamMessage methods
Sample_Project.zip 157	Reset 104
samples 156	Stop method 58, 81, 117, 133, 146
about 156	StreamMessage methods 98
building 158 , 159	Acknowledge 99
CRM 160	ClearBody 99
building 160	ClearProperties 99
configuring 162	GetProperty 100
creating the application 163	PropertyExists 100
database 160	ReadBoolean 101
running 166	ReadByte 101
directory structure 159	ReadBytes 101
environment 159	ReadChar 102
implementing 157	ReadDouble 102
running 164–166	ReadFloat 102
XA 160	ReadInt 103

ReadLong 103	CorrelationID 112
ReadObject 103	CorrelationIDAsBytes 112
ReadShort 104	DeliveryMode 112
ReadString 104	Destination 112
SetProperty 105	Expiration 112
WriteBoolean 105	MessageID 112
WriteBytes 106	Priority 112
WriteChar 106	Redelivered 112
WriteDouble 107	ReplyTo 112
WriteFloat 107	Text 112
WriteInt 107	Timestamp 112
WriteLong 108	Type 112 1
WriteObject 108	timestamp 21
WriteShort 108	Timestamp property 45, 60, 73, 98, 112
WriteString 109	TimeToLive property 78, 87, 120
StreamMessage object 72, 97	Topic methods 115
StreamMessage properties 97	ToString 116
CorrelationID 97	Topic object 115
CorrelationIDAsBytes 97	Topic properties 115
DeliveryMode 97	TopicName 115
Destination 98	Topic property 120, 130
Expiration 98	TopicConnection methods 116
MessageID 98	CreateTopicSession 116
Priority 98	Start 117
Redelivered 98	Stop 117
ReplyTo 98	TopicConnection object 116
Timestamp 98	TopicConnection properties 116
Type 98	ClientID 116
supporting documents 13	MetaData 116
supporting documents is	TopicConnectionFactory methods 118
_	CreateTopic 127
T	CreateTopicConnection 118
TemporaryQueue methods 110	CreateTopicConnectionEx 119
Temporary Queue object 109	TopicConnectionFactory object 118
Temporary Queue properties 109	TopicConnectionFactory properties 118
QueueName 109	HostName 118
TemporaryTopic methods 111	Port 118
Delete 110, 111	PortOffset 118
ToString 110, 111	TopicName property 111, 115
TemporaryTopic object 110	TopicPublisher methods 120
Temporary Topic properties 111	Publish 120
TopicName 111	TopicPublisher object 119
text conventions 13	TopicPublisher properties 119
Text property 112	DeliveryMode 119
TextMessage 22	DisableMessageID 120
TextMessage methods 113	DisableMessageTimestamp 120
Acknowledge 113	Priority 120
ClearBody 113	TimeToLive 120
ClearProperties 113	Topic 120
GetProperty 114	TopicRequestor methods 121
Property Exists 114	Close 121
SetProperty 115	Create 121
TextMessage object 72, 112	Request 122
TextMessage properties 112	TopicRequestor object 121
10. mileouge properties 114	

TopicSession method	WriteShort method 56, 108
CreateTemporaryTopic 127	WriteString method 109
TopicSession methods 123	WriteUTF method 57
Commit 123	
CreateBytesMessage 124	v
CreateDurableSubscriber 124	X
CreateMessage 125	XA compliance 25
CreatePublisher 125	XA sample 160
CreateStreamMessage 126	XAConnection object 27
CreateSubscriber 126	XAConnectionFactory object 27
CreateTextMessage 127, 151	XAQueueConnection methods 132
Recover 128	CreateQueueSession 132
Rollback 128	CreateXAQueueSession 133
Run 129	Start 133
Unsubscribe 129	Stop 133
TopicSession object 119, 122	XAQueueConnection object 27, 131
TopicSession properties 123	XAQueueConnection properties 132
MessageListener 123	ClientID 132
Transacted 123	MetaData 132
TopicSession property 149	XAQueueConnectionFactory method
TopicSubscriber methods 130	CreateQueueConnection 134
Close 130	XAQueueConnectionFactory methods 134
Receive 131	CreateQueueConnectionEx 135
ReceiveNoWait 131	CreateXAQueueConnection 135
TopicSubscriber object 129	CreateXAQueueConnectionEx 136
TopicSubscriber properties 130	XAQueueConnectionFactory object 27, 134
MessageListener 130	XAQueueConnectionFactory properties 134
MessageSelector 130	HostName 134
NoLocal 130	Port 134
Topic 130	PortOffset 134
ToString method 79 , 110 , 111 , 116	XAQueueSession methods 136
Transacted property 88, 93, 123, 136, 140, 149	Commit 137
Type property 45, 60, 73, 98, 112	CreateBytesMessage 137
	CreateMapMessage 137
11	CreateMessage 138
U	CreateStreamMessage 138
Unsubscribe method 129	CreateTextMessage 139
	Recover 139
*/	Rollback 139
V	Run 140
Visual Studio 158, 159	XAQueueSession object 27, 136
,	XAQueueSession properties 136
***	MessageListener 136
W	QueueSession 136
WriteBoolean method 53, 105	Transacted 136
WriteByte method 53, 105	XARecord object 27
WriteBytes method 54, 106	XASession methods 140
WriteChar method 54, 106	Commit 141
WriteDouble method 55, 107	CreateBytesMessage 141
WriteFloat method 55, 107	CreateMapMessage 141
WriteInt method 55, 107	CreateMessage 142
WriteLong method 56, 108	CreateStreamMessage 142
WriteObject method 56, 108	CreateTextMessage 142
,	Recover 143

Rollback 143

XASession object 27, 140

XASession Object methods

Run 144

XASession properties 140

MessageListener 140

Transacted 140

XATopicConnection methods 144

CreateTopicSession 145

CreateXATopicSession 145

Start 146

Stop 146

XATopicConnection object 27, 144

XATopicConnection properties 144

ClientID 144

MetaData 144

XATopicConnectionFactory methods 147

CreateTopicConnection 147

CreateTopicConnectionEx 148

CreateXATopicConnection 148

CreateXATopicConnectionEx 148

XATopicConnectionFactory object 27, 146

XATopicConnectionFactory properties 147

HostName 147

Port 147

PortOffset 147

XATopicSession methods 149

Commit 150

CreateBytesMessage 150

CreateMapMessage 150

CreateMessage 151

CreateStreamMessage 151

Recover 152

Rollback 152

Run 153

XATopicSession object 27, 149

XATopicSession properties 149

MessageListener 149

TopicSession 149

Transacted 149