SeeBeyond™ eBusiness Integration Suite

e*Gate API Kit Developer's Guide

Release 4.5.2

SEEBEYOND™

Version 20020507150805.

Contents

Chapter 1

Introduction

The e*Gate API kit enables you to create custom applications or modify existing external applications to interface with the e*Gate system. The API kit provides the following interfaces:

SeeBeyond Java Message Service (JMS)

- Java
- COM +

SeeBeyond Multiplexer (MUX) e*Way

- ActiveX
- C/C++
- Java
- Perl
- COBOL

1.1 Overview

The e*Gate API Kit provides two distinct IQ delivery service mechanisms:

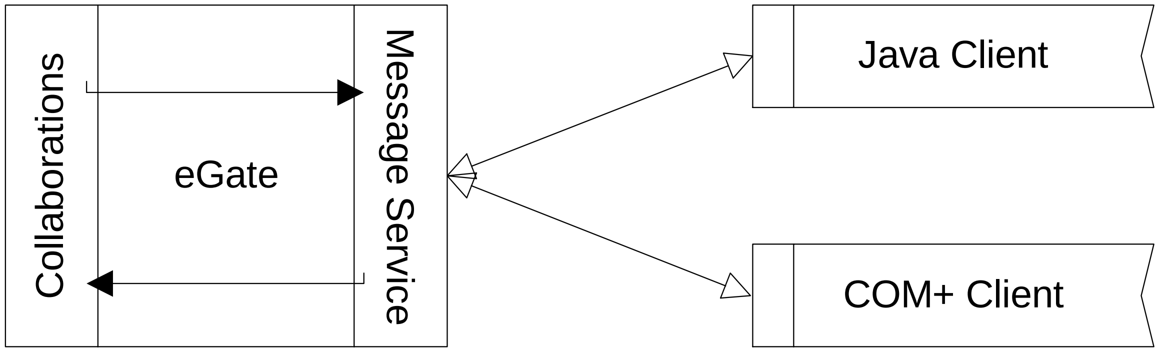
- The SeeBeyond Message Service
- The SeeBeyond Multiplexer e*Way

1.1.1 SeeBeyond Message Service Functionality

The SeeBeyond Message Service provides application with an API set for a common and elegant programming model, that is portable across messaging systems. Enterprise messaging systems are used to send notification of events and data between software applications. There are several common programming models supported by the SeeBeyond Message Service API: publish-and-subscribe, point-to-point, and request/reply, to name a few.

The diagram below shows the basic Message Service Data Flow.

Figure 1 Basic SeeBeyond Message Service Data Flow



Publish-and-subscribe

In a publish-and-subscribe scenario, one producer can send a single message to multiple consumers via a virtual channel called a **topic**. Consumers must subscribe to a topic to be able to receive it. Any messages addressed to a specific topic are delivered to all of that topic's consumers (subscribers). The pub/sub model is predominantly a push-based model, in that messages are automatically broadcast to consumers without them having to request or poll the topic for new messages.

Point-To-Point

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

Request-Reply

When the client sends a message and expects to receive a message in return, Request-Reply Messaging can be used. This is a synchronous object-messaging format. Request-reply uses either pub/sub or point-to-point to enable the functionality. JMS does not explicitly support Request-Reply Messaging, though it allows it in the context of the other methods.

Message Selector

Many messaging applications require the additional functionality of filtering and categorization of the messages they produce. If a message is sent to a single receiver, this can be done by including the criteria in the message, and the receiving client in turn, discards the ones not required. On the other hand, when a message needs to be distributed to many clients, including criteria into the message header, making it visible to the JMS provider, allows the provider to handle much of the filtering and routing, without impacting each client application.

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

Java Naming and Directory Interface

The Java Naming and Directory Interface (JNDI) provides naming and directory functionality to applications written using Java. JNDI consists of an API set along with a service provider interface (SPI). The Java applications use the JNDI API to access naming and directory services. The SPI allows the naming a directory services to be accessed transparently, thus providing the JNDI API access to their services.

JNDI is included in the Java 2 SDK, v 1.3 and later releases. It is also available as a Java Standard Extension for use with JDK 1.1 and Java 2 SDK, v1.2.

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

Compensating Resource Manager

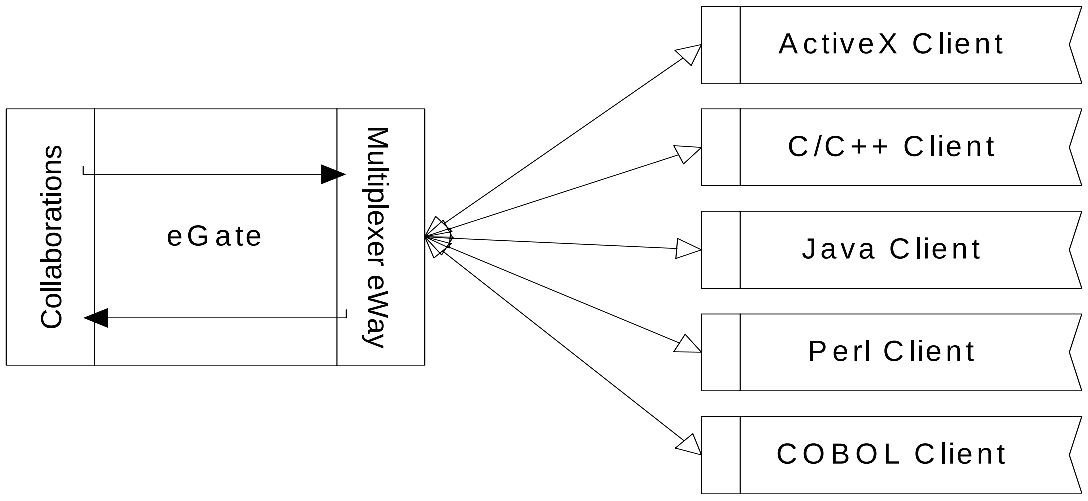
The Compensating Resource Manager (CRM) provides support for distributed transaction with multiple resource managers. These COM+ objects perform non-database operations as part of a distributed transaction. Distributed transaction involve multiple independent resource managers. If any part of the transactions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

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

1.1.2 SeeBeyond Multiplexer e*Way Functionality

The multiplexer provides support for both synchronous and asynchronous data transfer. The end-user also has the ability to perform real-time data queries and online transactions via back-office applications. This backend connectivity extends application, trading partner and business process integration to the worldwide web environment.

Figure 2 Basic SeeBeyond Multiplexer Data Flow



The e*Gate API kit Multiplexer supports three basic architectures.

Request-Reply

Data is sent to the e*Gate system and a response is returned. A client submits data (a **request)** to the e*Gate system. The e*Gate system processes the data as required. The e*Gate system returns data (a **reply/response)** to the same external application that submitted the request.

The e*Gate API kit uses a multiplexing e*Way that uses a proprietary IP-based protocol to multi-thread Event exchange between the e*Way and external systems or other e*Gate components.

Send-only

Using the same multiplexing e*Way component, data is sent to the e*Gate system but no data is returned

Receive

Receive, also known as Push-Port, an external system connects to the e*Gate system and allows for the delivery unsolicited Events from an external system, using the same multiplexing e*Way component.

1.2 Intended Reader

The reader of this guide is presumed to have the following responsibilities and possess these skill sets:

- Developer or System Administrator with responsibility for maintaining the e*Gate system.

- Have expert-level knowledge of Windows NT and UNIX operations and administration.
- Be thoroughly familiar with the programming and/or scripting language (C/C++, Java, Visual Basic, ASP, or Perl) in which the client component is written.
- Be thoroughly familiar with Windows-style GUI operations.

1.3 Supported Operating Systems

The Java Message Service is available on the following operating systems:

- Windows 2000, Windows 2000 SP1, and Windows 2000 SP2
- Windows NT 4.0 SP6a
- Solaris 2.6, 7, and 8
- AIX 4.3.3
- OS/390 V2R10 client only
- HP-UX 11.0 and HP-UX 11i
- Compaq Tru64 V4.0F and V5.0A
- AS/400 client only
- Japanese Windows 2000, Windows 2000 SP1, and Windows 2000 SP2
- Japanese Windows NT 4.0 SP6a
- Japanese Solaris 2.6, 7, and 8
- Japanese HP-UX 11.0
- Korean Windows 2000, Windows 2000 SP1, and Windows 2000 SP2
- Korean Windows NT 4.0 SP6a
- Korean Solaris 8
- Korean AIX 4.3.3
- Korean HP-UX 11.0

The Java Message Service COM+ APIs are available on the following operating system:

- Windows 2000, Windows 2000 SP1, and Windows 2000 SP2
- Japanese Windows 2000, Windows 2000 SP1, and Windows 2000 SP2
- Korean Windows 2000, Windows 2000 SP1, and Windows 2000 SP2

The MUX e*Gate API Kit is available on the following operating systems:

- Windows 2000, Windows 2000 SP1, and Windows 2000 SP2
- Windows NT 4 SP6a
- Solaris 2.6, 7, and 8

- AIX 4.3.3
- OS/390 V2R10
- HP-UX 11 and HP-UX 11i
- Compaq Tru64 V4.0F and V5.0A
- Japanese Windows 2000, Windows 2000 SP1, and Windows 2000 SP2
- Japanese Windows NT 4.0 SP6a
- Japanese Solaris 2.6, 7, and 8
- Japanese HP-UX 11.0
- Korean Windows 2000, Windows 2000 SP1, and Windows 2000 SP2
- Korean Windows NT 4.0 SP6a
- Korean Solaris 8

1.4 System Requirements

1.4.1 For Using Java Message Service APIs

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

- A TCP/IP network connection
- Java: Version 1.3.0 or higher
- A development environment with a compiler that is compatible with platforms supported by e*Gate. For example:
  - Sun Java Compiler 1.3.0 or higher

1.4.2 For Using Java Message Service COM+ APIs

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

- A TCP/IP network connection
- A development environment with a compiler that is compatible with platforms supported by e*Gate. For example:
  - Windows NT or Windows 2000: Microsoft Visual Basic

1.4.3 For Using MUX e*Gate APIs

To use the MUX e*Gate API Kit, you need the following:

- A TCP/IP network connection
- Java: Version 1.2.2 or higher

- A client system capable of executing an application that uses the e*Gate multiplexer APIs. The requirements for the client applications are as follows:
  - C/C++ software program with a compiler that is compatible with the platform supported by e*Gate. For example:
    - Windows NT/Windows 2000: Microsoft Visual C++ 6.0
    - UNIX: C Compiler or Sun C++
- Visual Basic or other application capable of using ActiveX components: The e*Gate libraries **stdole32.tlb** and **stdole2.tlb** must be installed on the client system. ActiveX support is available under Windows operating systems only.
- Perl: The following Perl libraries are supported:

OS	Perl
AIX 4.3.3	5.005_03 standard
HPUX 11.0, 11.i	5.005_03 HP-UX Developer Resource
Solaris 2.6, 7	download 5.005_03 from Sun Freeware
Solaris 2.8	5.005_03 standard
Linux Red Hat 6.2	5.005_03 standard
Compaq Tru64 5.0a	5.004_04
Windows NT/2000	5.6.1.629

HPUX clients also require that the Perl executable must be linked against the p-thread library (using the flag **-lpthread**) when it is built.

The above versions are the only versions that are officially supported and tested.

Note: *With the many compilers available, it is possible that some will not be compatible with the e*Gate environment.*

1.5 O/S 390 System Requirements

OS/390 system requirements and installation procedures are covered in Chapter 6, "Installation Instructions for OS/390" of the *e*Gate Integrator Installation Guide*.

OS/390 systems use the EBCDIC character set. As a consequence, ASCII-based systems cannot directly transport data to an EBCDIC-based system. ASCII to EBCDIC data conversion is necessary when data is sent from UNIX/Windows to OS/390. This data conversion should take place within a Collaboration.

To transport any EBCDIC data to an ASCII-based system (UNIX or Windows), you must first convert the data by using the ebcdic->ascii Monk function. Refer to the Monk Developer's Reference Guide for details about this function.

To use the COBOL portion of the e*Gate API Kit, you need the following:

1 An e*Gate Participating Host, version 4.5.1 or higher.

Server:

- IBM OS/390 or equivalent hardware
- Physical access CD-ROM
- TCP/IP connectivity
- Appropriate terminal for access to system

1.6 External System Requirements for OS/390

1.6.1 For Using CICS

To enable the e*Way to communicate properly with the Server system, the following are required:

- Cobol for OS/390 compiler must be available for use in the OS/390 Language Environment (LE), with the CICS TCP/IP socket elements available for inclusion in the link step. A DD statement pointing to the socket library should be added to the compile procedure (usually DFHYITVL).

  See below for a link to the IP CICS Sockets manual, which describes setup procedures:

  http://www-1.ibm.com/servers/s390/os390/bkserv/r10pdf/secureway.html

Note: *Select book #SC31-8518-01 to access the IP CICS Sockets Guide. This book explains the setup of TCP/IP Sockets for CICS, which is a requirement for the Cobol component of the e*Gate API Kit to function properly.*

  - OS/390 V2R10
  - Security package - install script RACF - ready
  - CICS 3.3 or higher or CICS TS 1.x
  - CICS TCP/IP socket interface must be installed and configuration for each region in which the Cobol API will be run.
  - COBOL for OS/390
  - Optional - Open Multiple Virtual System (OMVS) installed, configured, and operational.

1.6.2 For Using IMS

To enable the e*Way to communicate properly with the Server system, the following are required:

- Cobol for OS/390 compiler must be available for use in the OS/390 Language Environment (LE), with the MVS TCP/IP socket elements available for inclusion in the link step. A DD statement pointing to the socket library should be added to the compile procedure.

For additional information, consult the IBM website, document number SG24-5229-01, *"OS/390 eNetwork Communications Server TCP/IP Implementation Guide, Volume 3: MVS Applications"*

Note: *This book explains the setup of TCP/IP Sockets for MVS, which is a requirement for the IMS and Batch Cobol components of the e*Gate API Kit to function properly.*

- OS/390 V2R10
- Security package - install script RACF - ready
- IMS 6.1 or higher
- MVS TCP/IP socket interface must be installed, configured, and operational
- COBOL for OS/390
- Optional - Open Multiple Virtual System (OMVS) installed, configured, and operational.

1.6.3 For Using Batch

To enable the e*Way to communicate properly with the Server system, the following are required:

- Cobol for OS/390 compiler must be available for use in the OS/390 Language Environment (LE), with the MVS TCP/IP socket elements available for inclusion in the link step. A DD statement pointing to the socket library should be added to the compile procedure.

For additional information, consult the IBM website, document number SG24-5229-01, *"OS/390 eNetwork Communications Server TCP/IP Implementation Guide, Volume 3: MVS Applications"*

Note: *This book explains the setup of TCP/IP Sockets for MVS, which is a requirement for the IMS and Batch Cobol components of the e*Gate API Kit to function properly.*

- OS/390 V2R10
- Security package - install script RACF - ready
- MVS TCP/IP socket interface must be installed, configured, and operational
- COBOL for OS/390
- Optional - Open Multiple Virtual System (OMVS) installed, configured, and operational.

Chapter 2

Installing the e*Gate API Kit

This chapter describes the procedures necessary to install the e*Gate API Kit from the e*Gate installation CD-ROM.

After the product is installed, you must customize it to execute your site-specific business logic and to interact with your other systems as required.

2.1 Supporting Documents

The following documents are designed to work in conjunction with the *e*Gate API Kit User's Guide* and to provide additional information that may prove useful to you.

- *e*Gate Integrator Installation Guide.*
- *e*Gate Integrator System Administration and Operations Guide*.
- *e*Gate Integrator User's Guide.*
- *SeeBeyond JMS Intelligent Queue User's Guide*
- *SeeBeyond Master Index (SeeBeyond_Index.pdx; refer to e*Gate Integrator User's Guide for instructions on how to access.)*
- *README.txt* file on the e*Gate installation CD ROM.

2.2 Windows NT and 2000

Before installing the e*Gate API Kit, please read the following sections to ensure a smooth and error-free installation.

You must have Administrator privileges to successfully install e*Gate.

The installation of the e*Gate API Kit must be installed after successfully completing the installation of the e*Gate Registry Host. For more information about installing the Registry Host, see the e*Gate Integrator Installation Guide.

2.2.1 Pre-installation

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

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

2.2.2 Installing the e*Gate API Kit

To install the e*Gate API Kit on a Windows system:

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

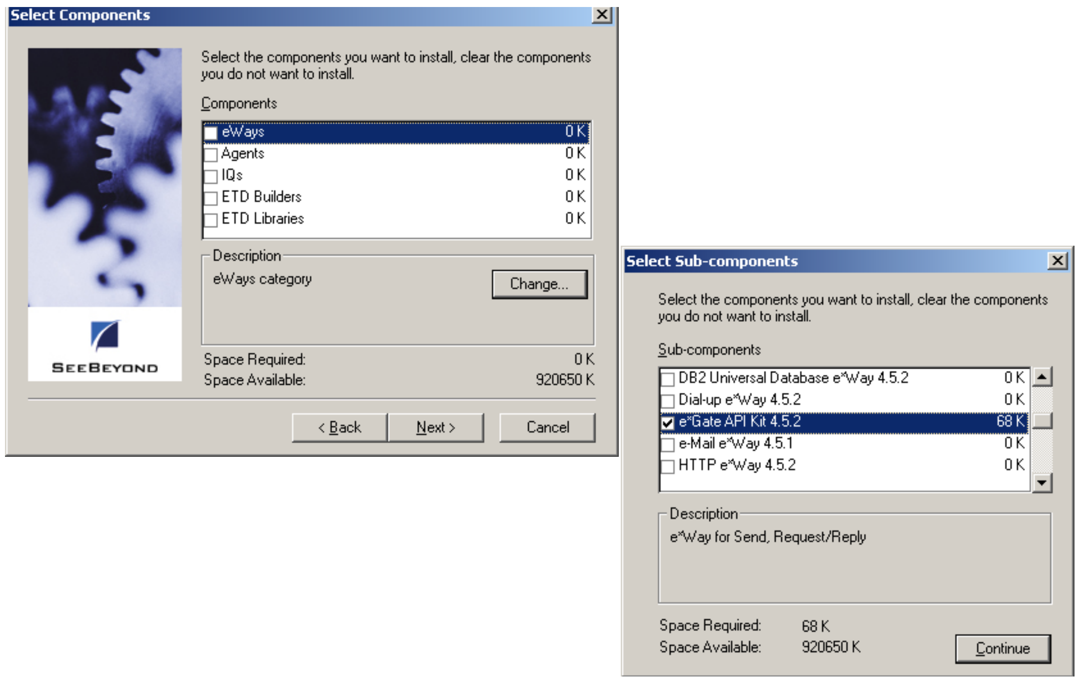
2 Insert the e*Way installation CD-ROM into the CD-ROM drive.

3 If the CD-ROM drive's "Autorun" feature is enabled, the setup application should launch automatically; skip ahead to step 4. Otherwise, use the Windows NT Explorer or the Control Panel's **Add/Remove Applications** feature to launch the file **setup.exe** on the CD-ROM drive.

4 The InstallShield setup application launches.

5 When the Select Components screen appears, click the Change button to select the e*Gate API Kit.

Figure 3 Select Components



Select Components

Select the components you want to install, clear the components you do not want to install.

Components

☐ eWays	0 K
☐ Agents	0 K
☐ IQs	0 K
☐ ETD Builders	0 K
☐ ETD Libraries	0 K

Description
eWays category

Space Required: 0 K
Space Available: 920650 K

Select Sub-components

Select the components you want to install, clear the components you do not want to install.

Sub-components

☐ DB2 Universal Database e*Way 4.5.2	0 K
☐ Dial-up e*Way 4.5.2	0 K
✓ e*Gate API Kit 4.5.2	68 K
☐ e-Mail e*Way 4.5.1	0 K
☐ HTTP e*Way 4.5.2	0 K

Description
e*Way for Send, Request/Reply

Space Required: 68 K
Space Available: 920650 K

6 Follow the on-screen instructions to install the e*Way.

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

2.3 Unix

2.3.1 Pre-installation

Before installing the e*Way on your UNIX system, please read the following sections to ensure a smooth and error-free installation.

You will need regular (non-root) user access to begin the e*Gate installation.

2.3.2 Installing the e*Gate API Kit

To install the e*Gate API Kit on a UNIX system:

1. Log in on the workstation containing the CD-ROM drive, and insert the CD-ROM into the drive.
2. If necessary, mount the CD-ROM drive.
3. At the shell prompt, type

   ```
   cd  /cdrom
   ```
4. Start the installation script by typing:

   ```
   setup.sh
   ```
5. Follow the prompts to accept user license information etc.
6. A menu of options opens. Enter the number corresponding to the "e*Gate Add-on Applications" option (1). Then, follow any additional on-screen directions.
7. Enter the number corresponding to "eWays" (1)
8. Enter the number corresponding to the "e*Gate API Kit" (number may vary).
9. Be sure to install support for any additional platform support.

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

2.4 Directories Created by the Installation

The e*Gate API Kit installation process installs the following files within the e*Gate directory tree. Files are installed within the "egate\client" tree on the Participating Host and committed to the "default" schema on the Registry Host.

Table 1 Files Required for External Application Support

Location	File	General Purpose
eGate/client/classes eGate/client/classes/thirdparty/sun eGate/client/classes/thirdparty/sun	stcjms.jar jms.jar jta.jar	JMS Java clients XA support
eGate/client/bin eGate/client/bin eGate/client/bin eGate/client/bin	stc_mscom.dll stc_msclient.dll stc_mscommon.dll stc_msapi.dll	JMS COM+ clients and CRM
eGate/client/bin eGate/client/bin eGate/client/bin eGate/client/bin	stc_msclient.dll stc_mscommon.dll stc_msapi.dll stcms.exe	JMS Server
eGate/client/bin	stc_ewipmpclnt.dll stc_common.dll	MUX C and C++ clients
eGate/client/bin eGate/client/bin eGate/client/bin	stc_xipmpclnt.dll stc_common.dll stc_ewipmpclnt.dll	MUX ActiveX clients
eGate/client/perl eGate/client/bin	stc_ewipmpclntperl.pm stc_ewipmpclntperl.dll stc_common.dll	MUX Perl clients
eGate/client/classes	stcph.jar	MUX Java clients
eGate/client/bin eGate/client/configs/stcewipmp	stcewipmp.exe stc_common.dll stc_ewipmpclnt.dll stcewipmp.def	Required for all MUX server.

2.5 OS/390

The installation tape contains the data sets listed in Table 2.

Table 2 Installation Tape Data sets

Dataset Name	Contents
STC.RESTORE.JCL	Physical Sequential Datasets containing the JCL for this tape.
STC.JCLLIB	Partition Data set that contains installation jobs and control cards.
STC.LOADLIB	Load Library that contains the MUXAPI, MUXIMS, MUXBAT load modules.
STC.MUX.OBJECT	Object library containing the object modules.
STC.MUX.UNLOAD	Unload data set containing object.

2.5.1 Copying the Tape Contents to Disk

Create and submit the following job to copy the load library to disk:

```
// JOB CARD
// TAPECOPY PROC PREFIX=custpref, <== CUSTOMER HIGH LEVEL QUALIFIER
//       LOADBLK=TRK, <== BLOCKING FACTORY FOR LOAD LIBRARY
//       LOADPRI=45,  <== PRIMARY ALLOCATION FOR LOAD LIBRARY
//       LOADSEC=15,  <== SECONDARY ALLOCATION FOR LOAD LIBRARY
//       LOADDIR=10,  <== DIRECTORY BLOCKS FOR LOAD LIBRARY
// IEBCOPY EXEC PGM=IEBCOPY
// SYSPRINT DD SYSOUT=*
// *
// * COPY MUXAPI LOAD LIBRARY TO DISK
// *
// INDD1 DD DSN=STC.LOADLIB,DISP=OLD,UNIT=CART,
//       VOL=(,RETAIN,SER=STC390),LABEL=(1,SL)
// OUTDD1 DD DSN=&PREFIX..STC.LOADLIB,DISP=(NEW,CATLG,DELETE),
//       UNIT=SYSDA,
//       SPACE=(&LOADBLK,(&LOADPRI,&LOADSEC,&LOADDIR))
// SYSIN DD DUMMY
//
```

Linking the COBOL API Load Models

To link the COBOL API in the link step of the compile for the calling program, do the following:

```
//LKED.SYSINN DD *
//  INCLUDE SYSLIB(MUXxxx)
//  ...
//  NAME ...
/*
```

Note: *Where MUXxxx is MUXAPI for CICS, MUXIMS for IMS, and MUXBAT for Batch.*

2.5.2 Verifying the CICS Transaction Server Environment for e*Gate

For CICS only: To verify CICS Sockets Support and Language Environment is enabled, look for the following messages at the CICS startup:

CICS Sockets Initialization:

```
DFHSO0100i applid Sockets domain initialization has started.
DFHSO0101I applid Sockets domain initialization has ended.
```

Language Environment Initialization:

```
DFHAP1203I applid Language Environment/370 is being initialized.
```

You should not see:

```
DFHAP1200 applid A CICS request to the Language Environment/370 has
failed. Reason code rc.
```

Chapter 3

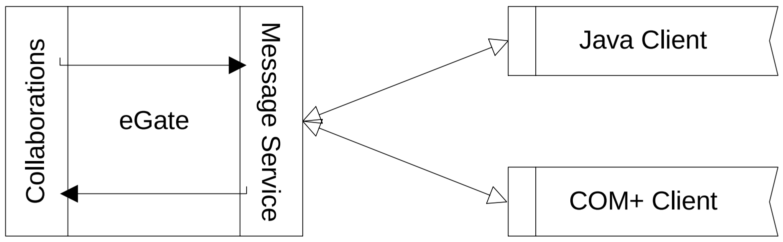
Configuring the Message Service

This chapter explains how to configure the three separate components that constitute SeeBeyond's implementation of the Java Message Service:

- Message Service Client: the external application
- Message Server: the data container and router
- e*Way Connection: the link between e*Gate and the external system

The following diagram illustrates the communication between each component.

Figure 4 Message Service Communication Architecture

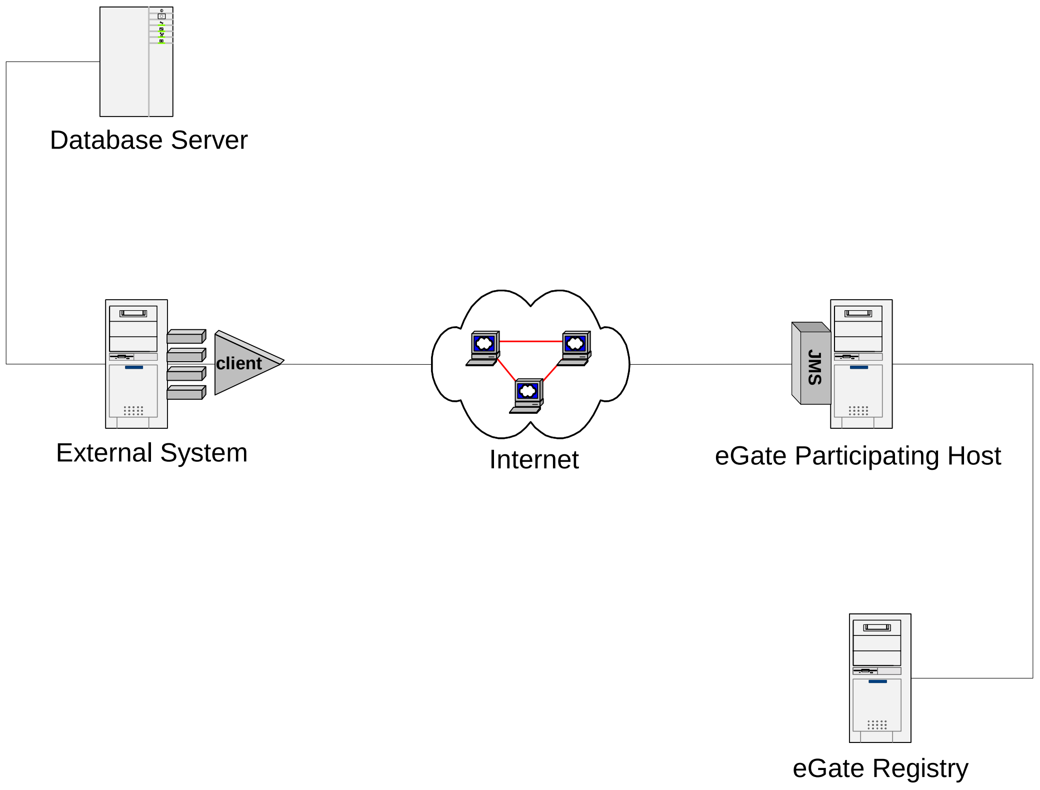


3.1 Configuring the Message Service Clients

The current SeeBeyond Message service supports both Java and COM+ clients. The sections that follow provide the information necessary to configure both of these clients.

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

Figure 5 Message Service Communication Architecture



In some form, the following components must exist:

- e*Gate Registry Host (e*Gate Server)
- e*Gate Participating Host (e*Gate Client)
- External System (SeeBeyond 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 e*Gate Participating Host and the External System may exist on one physical machine.*

3.1.1 Java Client

Once the e*Gate API Kit has been installed successfully, additional steps are required to run Java JMS client programs. Both the Java client, which represents the machine where the external code resides, and the Java server, which represents the machine where the Message Server (also referred to as the SeeBeyond JMS IQ Manager) resides requiring handling. In this section, the setup steps are included for setting up the Message Service to use Java.

Setting up the Java Client

To begin using the Message Service for Java, do the following:

Copy the **`jms.jar`** and **`jta.jar`** files from `eGate\client\ThirdParty\sun` directory to a directory on your external system. Copy the **`stcjms.jar`** file from `eGate\client\classes` directory to the same directory on your external system that you copied the **`jms.jar`** and **`jta.jar`** files to.

Modify the CLASSPATH on your external system to include the **jms.jar** and **stcjms.jar** files. For XA support, you will also need to include the **jta.jar** in your path.

To use the Java APIs, you need the following import statements in your Java files.

```
import javax.jms.*;
import com.seebeyond.jms.*;
```

3.1.2 COM+ Client

Once the e*Gate API Kit has been installed successfully, additional steps are required to finish the setup, before data exchange can begin. Both the COM+ client, which represents the machine where the external code resides, and the Java server, which represents the machine where the Message Server (also referred to as the SeeBeyond JMS IQ Manager) resides requiring handling. In this section, the setup steps are included for setting up the Message Service to use COM+.

Setting up the COM+ Client

For all COM+ implementations, to begin using the Message Service for COM+, do the following:

Copy the **stc_mscom.dll**, **stc_msclient.dll**, **stc_mscommon.dll**, **stc_msapi.dll** files from your egate\client\bin to a directory on your external system.

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

regsvr32 your_path_location\stc_mscom.dll

Viewing the Message Service COM+ APIs Using Microsoft Visual Basic 6.0

You may view the JMS COM+ APIs using any application that is capable of viewing COM+ APIs.

To begin viewing the APIs:

1 Open Microsoft Visual Basic 6.0
2 From the New Project dialog box, click Standard EXE and click Open.
3 From the Project toolbar, click References...
4 From the References dialog box, select SeeBeyond Message Service 1.0.
5 Click OK.

6 From the View toolbar, click Object Browser.

7 From the <All Libraries> list box, select STC_MSCOM.

8 Press the F2 button, to open the Object Browser dialog box.

9 From the <All Libraries> drop down button, select STC_MSCOM to view the supported classes and methods.

10 Highlight the class to view the member methods and properties.

Compensating Resource Manager (CRM)

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

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

Compensating Resource Manager (CRM) Setup

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

1. Register **stc_mscom.dll** by performing the following:

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

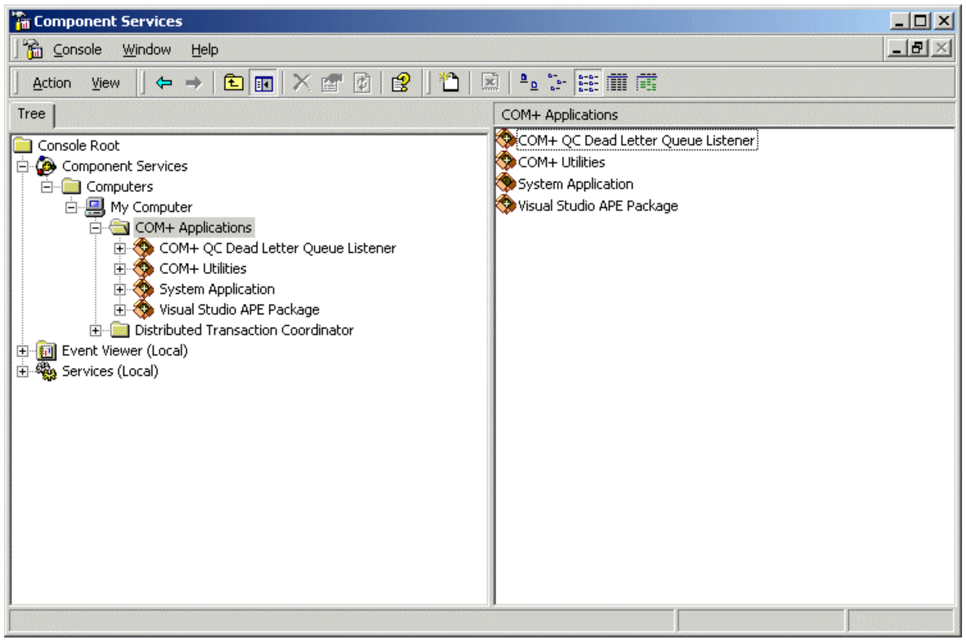
```
regsvr32 your_path_location\stc_mscom.dll
```

2 From the following location:

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

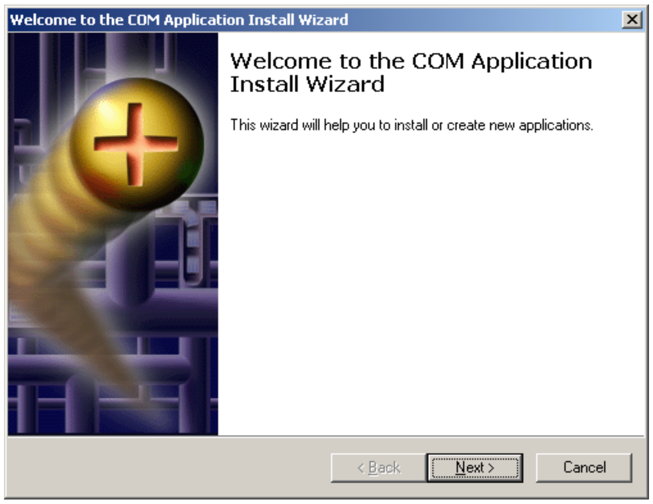
Expand the Component Services folder. Right click on Com+ Applications.

Figure 6 Component Services Folder



3 Select New\Application. The COM Application Install Wizard opens. Click Next to continue.

Figure 7 COM Application Install Wizard

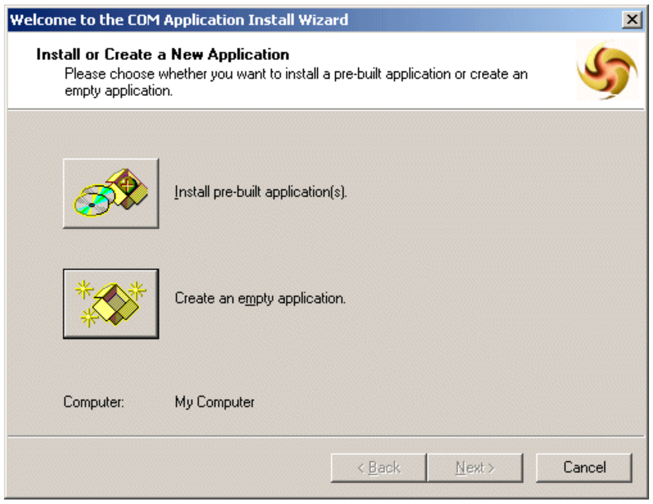


Welcome to the COM Application Install Wizard

This wizard will help you to install or create new applications.

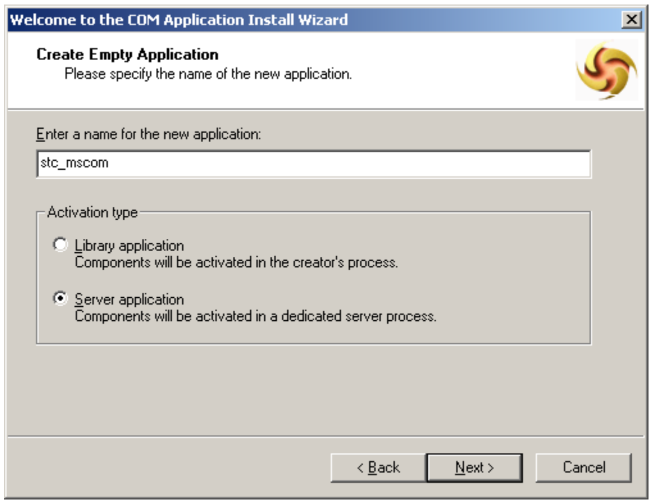
4 Select create an empty application.

Figure 8 COM Application Install Wizard


Welcome to the COM Application Install Wizard
Install or Create a New Application
Please choose whether you want to install a pre-built application or create an empty application.
Install pre-built application(s).
Create an empty application.
Computer:
My Computer
< Back
Next >
Cancel

5 Enter **stc_mscom** for the name of the application, and select Server Application for the Application Type.

Figure 9 COM Application Install Wizard: New Application


Welcome to the COM Application Install Wizard
Create Empty Application
Please specify the name of the new application.
< Back
Next >
Cancel

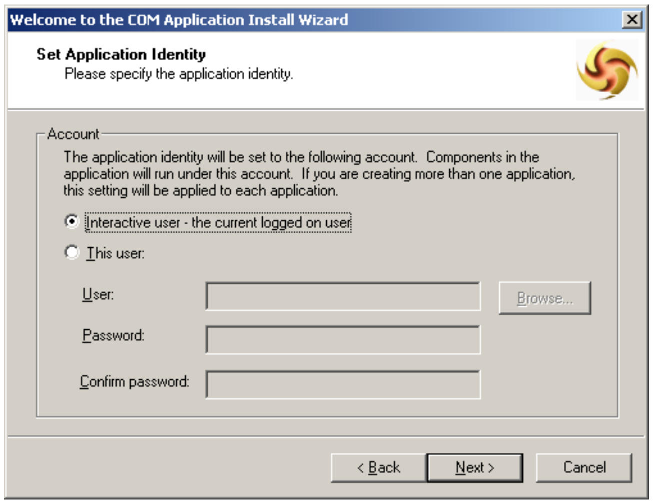
Enter a name for the new application: stc_mscom

Activation type

- ○ Library application
  Components will be activated in the creator's process.
- ● Server application
  Components will be activated in a dedicated server process.

6 Set the Application Identity to Interactive User. Click Next.

Figure 10 COM Application Install Wizard: Set Application Identity



Welcome to the COM Application Install Wizard

Set Application Identity
Please specify the application identity.

Account

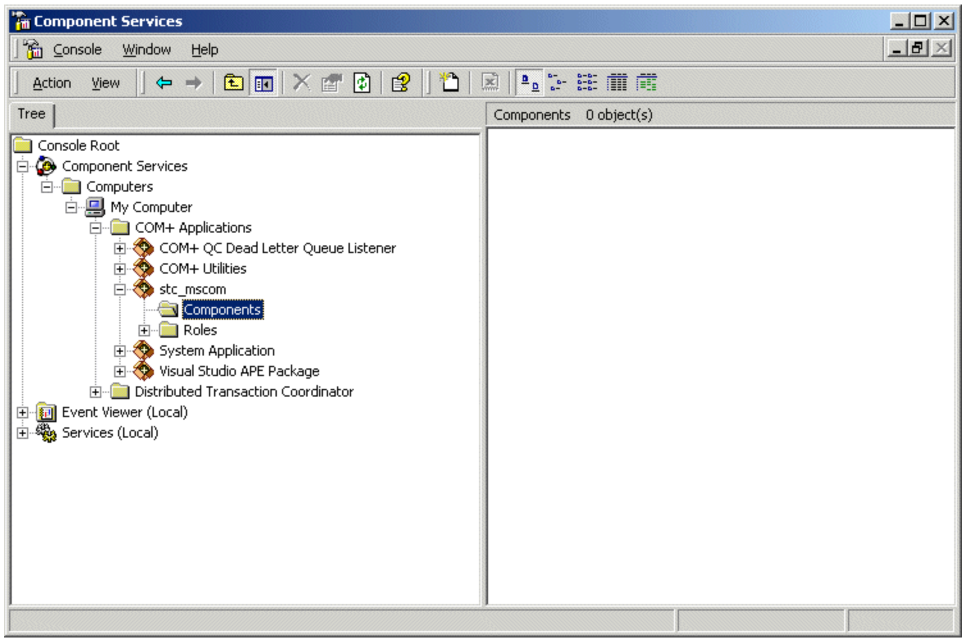
The application identity will be set to the following account. Components in the application will run under this account. If you are creating more than one application, this setting will be applied to each application.

[✓ Interactive user - the current logged on user]
This user:
User:
Password:
Confirm password:

7 Click Finish.

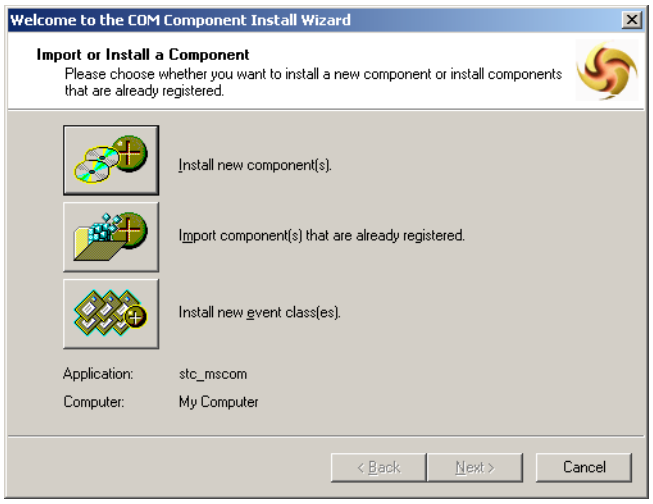
8 Expand **stc_mscom** component, then, right click on the Components folder, select New Component.

Figure 11 Component Services: stc_mscom Component



9 The COM Component Install Wizard opens. Select Install new components to continue.

Figure 12 COM Component Install Wizard


Welcome to the COM Component Install Wizard
Install new component(s).
Import component(s) that are already registered.
Install new event class(es).
< Back
Next >
Cancel

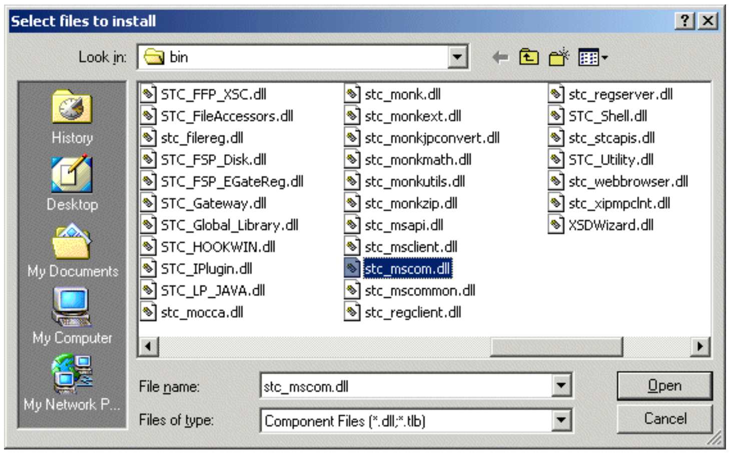
Import or Install a Component
Please choose whether you want to install a new component or install components that are already registered.

Application: stc_mscom
Computer: My Computer

10 Click Add to navigate to the location of the **stc_mscom.dll**. If you are running the **.dll** on the same machine that e*Gate was installed, the file is located:

```
<eGate>\client\bin\
```

Figure 13 COM Component Install Wizard



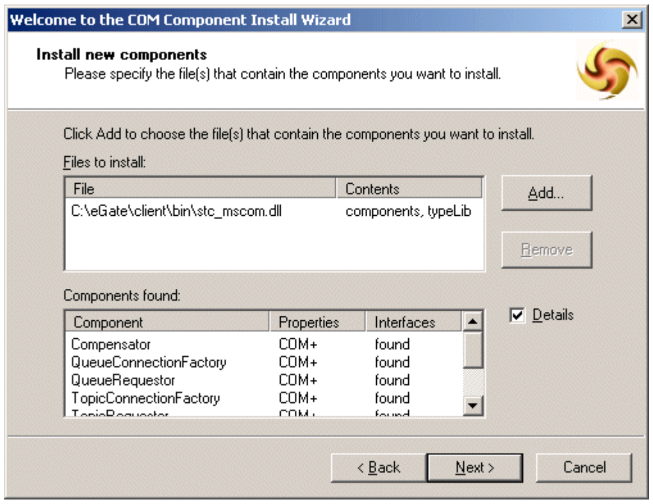
Look in: bin

STC_FFP_XSC.dll, STC_FileAccessors.dll, stc_filereg.dll, STC_FSP_Disk.dll, STC_FSP_EGateReg.dll, STC_Gateway.dll, STC_Global_Library.dll, STC_HOOKWIN.dll, STC_IPlugin.dll, STC_LP_JAVA.dll, stc_mocca.dll, stc_monk.dll, stc_monkext.dll, stc_monkjpconvert.dll, stc_monkmath.dll, stc_monkutils.dll, stc_monkzip.dll, stc_msapi.dll, stc_msclient.dll, stc_mscom.dll, stc_mscommon.dll, stc_regclient.dll, stc_regserver.dll, STC_Shell.dll, stc_stcapis.dll, STC_Utility.dll, stc_webbrowser.dll, stc_xipmpclnt.dll, XSDWizard.dll

File name: stc_mscom.dll
Files of type: Component Files (*.dll;*.tlb)

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

Figure 14 COM Component Install Wizard: Add



Welcome to the COM Component Install Wizard

Install new components
Please specify the file(s) that contain the components you want to install.

Click Add to choose the file(s) that contain the components you want to install.

Files to install:

File	Contents
C:\eGate\client\bin\stc_mscom.dll	components, typeLib

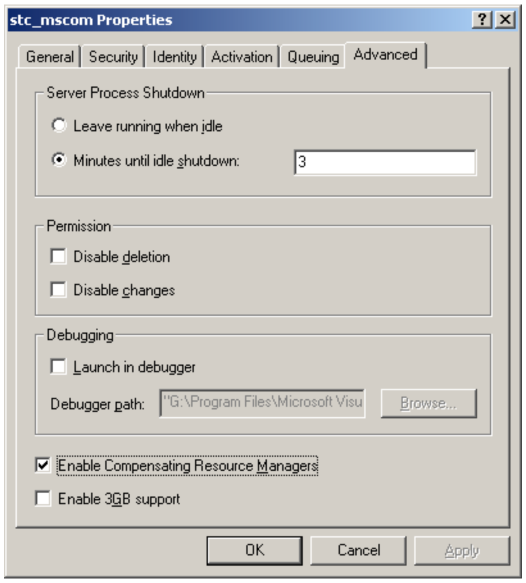
Components found:

Component	Properties	Interfaces
Compensator	COM+	found
QueueConnectionFactory	COM+	found
QueueRequestor	COM+	found
TopicConnectionFactory	COM+	found

The components appear in the Components found dialog box, ensure that the Show Details box is selected. Click Next to continue. Click Finish.

11 Select the **stc_mscom** component, right click, select properties. From the Advanced tab, enable the Compensating Resource Manager (CRM).

Figure 15 stc_mscom Properties: Advanced



stc_mscom Properties

Server Process Shutdown
- ○ Leave running when idle
- ● Minutes until idle shutdown: 3

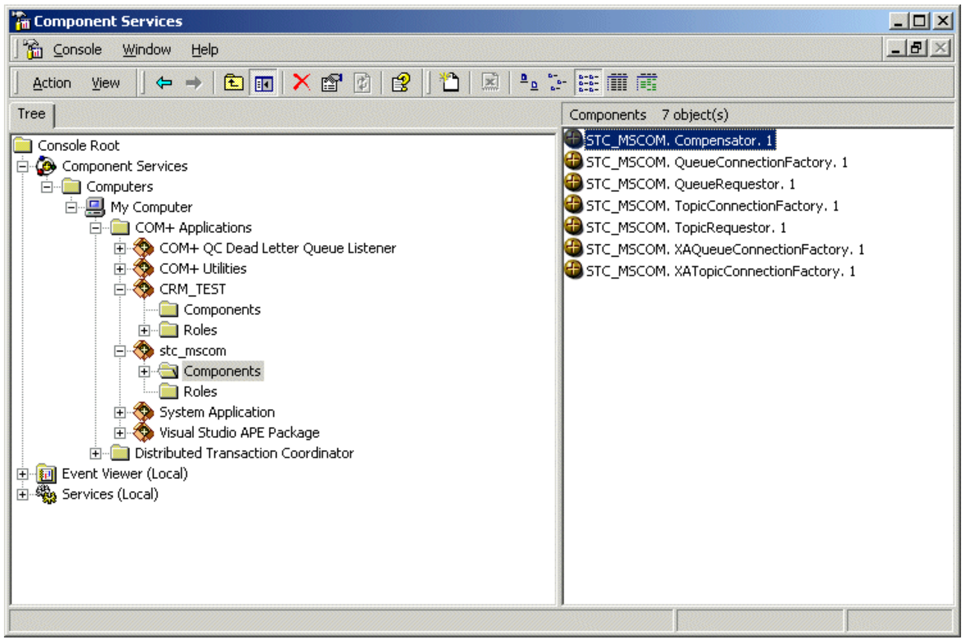
Permission
- ☐ Disable deletion
- ☐ Disable changes

Debugging
- ☐ Launch in debugger
- Debugger path: "G:\Program Files\Microsoft Visu

- ☑ Enable Compensating Resource Managers
- ☐ Enable 3GB support

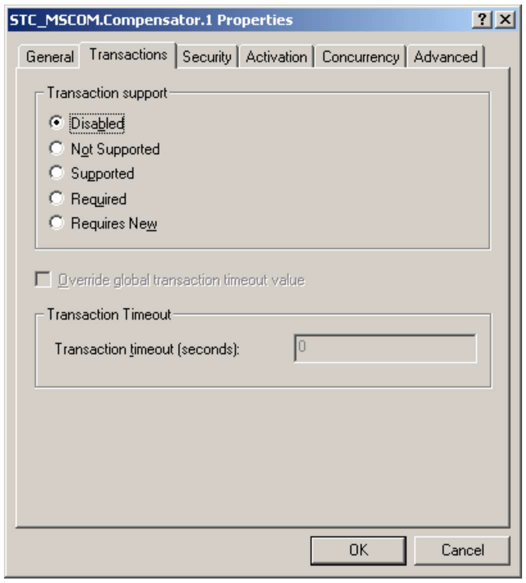
12 Expand the **stc_mscom** component. Select the Components folder, right click. Right click on **STC_MSCOM.Compensator**, select properties.

Figure 16 STC_MSCOM.Compenstator Properties


Component Services
Console Window Help
Action View
Tree
Console Root
Component Services
Computers
My Computer
COM+ Applications
COM+ QC Dead Letter Queue Listener
COM+ Utilities
CRM_TEST
Components
Roles
stc_mscom
Components
Roles
System Application
Visual Studio APE Package
Distributed Transaction Coordinator
Event Viewer (Local)
Services (Local)
Components 7 object(s)
STC_MSCOM. Compensator. 1
STC_MSCOM. QueueConnectionFactory. 1
STC_MSCOM. QueueRequestor. 1
STC_MSCOM. TopicConnectionFactory. 1
STC_MSCOM. TopicRequestor. 1
STC_MSCOM. XAQueueConnectionFactory. 1
STC_MSCOM. XATopicConnectionFactory. 1

13 From the Transactions tab, disable Transaction support.

Figure 17 STC_MSCOM.Compenstator Properties:Transaction Support


STC_MSCOM.Compensator.1 Properties

General | Transactions | Security | Activation | Concurrency | Advanced

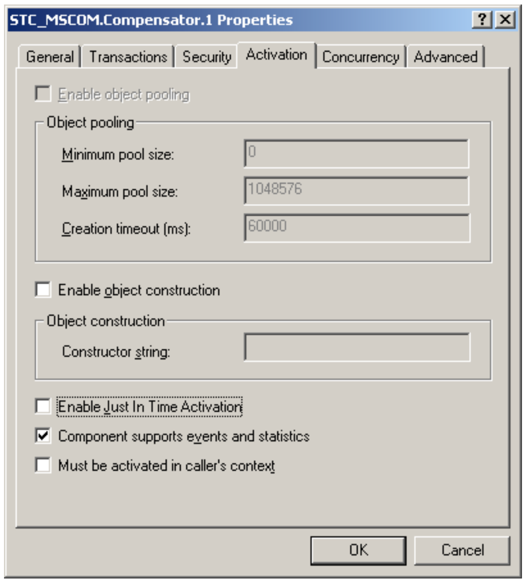
Transaction support
- [✓ Disabled]
- Not Supported
- Supported
- Required
- Requires New

Override global transaction timeout value

Transaction Timeout
Transaction timeout (seconds): 0

14 From the Activation tab, disable Just In Time Activation.

Figure 18 STC_MSCOM.Compenstator Properties:Activation



Enable object pooling

Object pooling

Minimum pool size: 0

Maximum pool size: 1048576

Creation timeout (ms): 60000

Enable object construction

Object construction

Constructor string:

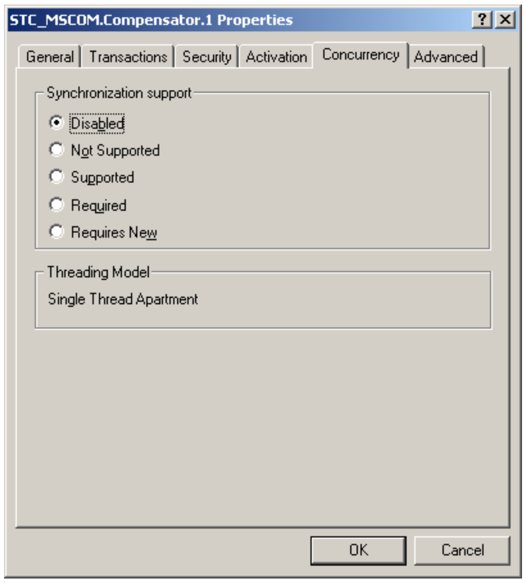
Enable Just In Time Activation

✓ Component supports events and statistics

Must be activated in caller's context

15 From the Concurrency tab, disable synchronization support. Click OK.

Figure 19 STC_MSCOM.Compenstator Properties:Concurrency



Synchronization support

● Disabled

○ Not Supported

○ Supported

○ Required

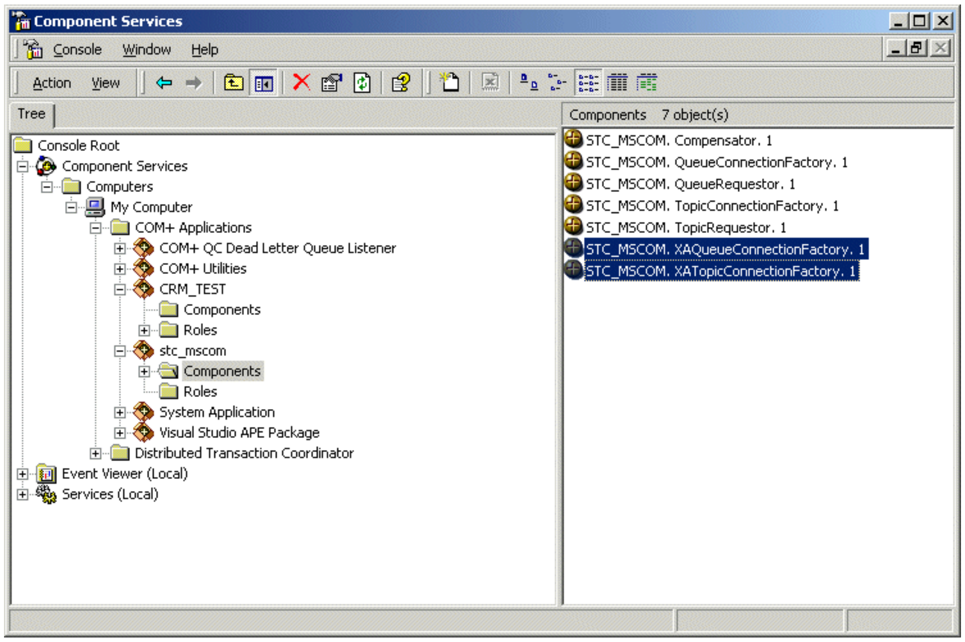
○ Requires New

Threading Model

Single Thread Apartment

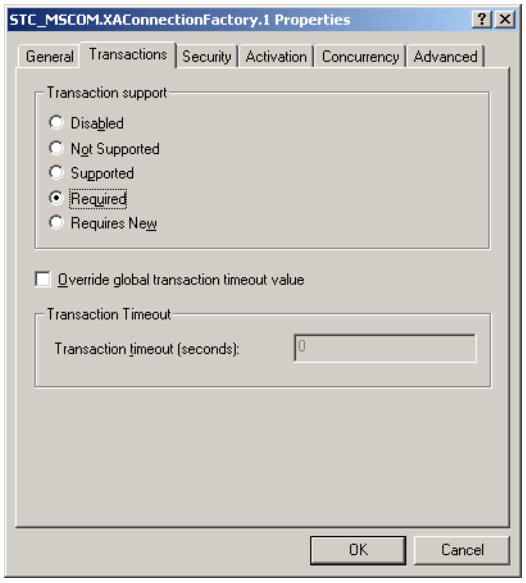
16 From the components folder, right click on **STC_MSCOM.XAQueueConnectionFactory**, and **STC_MSCOM.XATopicConnectionFactory**, select properties.

Figure 20 STC_MSCOM.XAConnectionFactory Properties


Component Services
Console Window Help
Action View
Tree
Console Root
Component Services
Computers
My Computer
COM+ Applications
COM+ QC Dead Letter Queue Listener
COM+ Utilities
CRM_TEST
Components
Roles
stc_mscom
Components
Roles
System Application
Visual Studio APE Package
Distributed Transaction Coordinator
Event Viewer (Local)
Services (Local)
Components 7 object(s)
STC_MSCOM. Compensator. 1
STC_MSCOM. QueueConnectionFactory. 1
STC_MSCOM. QueueRequestor. 1
STC_MSCOM. TopicConnectionFactory. 1
STC_MSCOM. TopicRequestor. 1
STC_MSCOM. XAQueueConnectionFactory. 1
STC_MSCOM. XATopicConnectionFactory. 1

17 From the Transactions tab, require Transaction support.

Figure 21 STC_MSCOM.XAConnectionFactory Properties:Transaction Support


STC_MSCOM.XAConnectionFactory.1 Properties

General | Transactions | Security | Activation | Concurrency | Advanced

Transaction support
- Disabled
- Not Supported
- Supported
- [✓ Required]
- Requires New

☐ Override global transaction timeout value

Transaction Timeout
Transaction timeout (seconds): 0

18 From the Activation tab, enable Just In Time Activation.

19 From the Concurrency tab, require Synchronization support. Click OK.

3.2 Configuring the Message Server

For information about the architecture and specific operation of the JMS IQ Manager (Message Server), see the *SeeBeyond JMS Intelligent Queue User's Guide.*

The SeeBeyond JMS IQ Manager is compliant with JMS version 1.0.2, and provides persistent nonvolatile storage of messages (Events), along with the necessary routing. The SeeBeyond JMS IQ Manager acts as a Message Server.

3.2.1 Considerations

The JMS Topic/Queue names and the e*Gate Event Types must coincide.

The individual writing any external JMS code must know the expected data format (byte or text), the name of the Topic/Queue (which must coincide with the Event Type name), the name of host and port number of the JMS client.

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

3.2.2 JMS IQ Manager Configuration Parameters

For more information about the JMS IQ Managers, see the *SeeBeyond JMS Intelligent Queue User's Guide*.

This section describes the configuration parameters and the external configuration requirements for the SeeBeyond JMS IQ Manager.

Configuration parameters are set using the IQ Manager Editor.

To change IQ Manager configuration parameters:

1. In the Enterprise Manager's Component editor, select the IQ Manager you want to configure and display its properties.
2. Ensure that the IQ Manager type is set to SeeBeyond JMS on the drop-down menu.
3. Under **Configuration File**, click **New** to create a new file, **Find** to select an existing configuration file, or **Edit** to edit the currently selected file.
4. In the **Additional Command Line Arguments** box, type any additional command line arguments that the may be required, taking care to insert them *at the end* of the existing command-line string. Be careful not to change any of the default arguments unless you have a specific need to do so.

The configuration parameters are organized into the following sections:

- **DB Settings** on page 39
- **Message Settings** on page 41
- **Server Settings** on page 43
- **Topic Settings** on page 43
- **Trace Settings** on page 43

Note: *The JMS term "topic" is used interchangeably with the e*Gate term "Event Type"; the JMS term "message" is used interchangeably with the e*Gate term "Event"; and the term "Server" is used generically for "SeeBeyond JMS IQ Manager."*

DB Settings

The **DB Settings** parameters govern the persistent message store, transaction-log files, and the disk write memory cache.

DBPath

Description

Specifies the directory in which the persistent message store and transaction-log files reside.

Required Values

A string.

If specified as a fully qualified path to the Message Service data directory, a variable is assigned to **.egate.store MessageServiceData**. If blank, the Message Service uses the value of the **.egate.store MessageServiceData** variable, as the base data directory. It then creates a subdirectory under that directory named with the UID of the IQManager to store the data files. The format uses the **forward** slash (/) as a path delimiter.

On Windows, **.egate.store** normally contains the line:

```
MessageServiceData=C:\EGATE\Client\stcms
```

If you keep all defaults, the persistent store files will reside in:

```
C:\EGATE\Client\stcms\IQManagerUID
```

(where *IQManagerUID* is replaced by the actual UID of the IQ Manager)

On UNIX, **.egate.store** normally contains the line:

```
MessageServiceData=/usr/egate/client/stcms
```

If you keep all defaults, the persistent store files will reside in:

```
/usr/egate/client/stcmsIQManagerUID
```

(where *IQManagerUID* is replaced by the actual UID of the IQ Manager)

If a relative path is specified, the Message Service overrides the **.egate.store MessageServiceData** setting. The Message Service stores the data files in the directory specified by **DBPath** relative to the directory containing the **stcms.exe** executable (usually eGate/client/bin).

If specified as an absolute path, the Message Service overrides the **.egate.store MessageServiceData** setting. The Message Service stores the data files in the directory specified by **DBPath**.

DBSuffix

Description

Specifies the characters to use as a file extension for the file names of the persistent message store and transaction-log files.

Required Values

A string. The default is **dbs**

Note: *If you keep all defaults, the persistent store files will have the form C:\eGate\client\stcms\stcms#####.**dbs**.*

DBCacheSize

Description

Specifies the total number of pages in the database system disk cache. A larger cache means better performance for more active data files.

Required Values

An integer between 1 and 999999999. The default is 1024. (A page is 512 bytes on Windows, 1024 bytes on UNIX.) Range: **1 - DBSegmentSize**

DBSegmentSize

Description

Specifies the total number of pages in a single DB file. (A page is 512 bytes on Windows, 1024 bytes on UNIX.)

Required Values

An integer between 1 and 999999999. The default is 16384.

Limits:

Set this to at least **(2 * the total number of anticipated durable subscribers).**

Note: ***DBSegmentSize** must be set greater than the size of the largest anticipated message. Allow a generous margin of error.*

DBMinSegments

Description

Specifies the minimum number of files, of size **DBSegmentSize**, initially created and maintained by the Server for its persistent message store and transaction log. When the minimum is exceeded, the Server allocates additional segments on an as-needed basis, up to the maximum set by **DBMaxSegments**.

Required Values

An integer between **1** and **99999**. The default is 4.

Limits:

Must be **>= 1**, up to **99999.**

DBMaxSegments

Description

Specifies the maximum total number of files that the Server will create and maintain for its persistent message store and transaction log. This effectively limits the amount of disk space that the Server will use. If the Server needs to write data that would exceed this limit, it exits gracefully and outputs an appropriate error message to the trace log.

The SeeBeyond JMS IQ Manager should not be used as a semi-permanent storage medium without sufficient memory and disk resources. To control the memory and disk resources needed by the Server, use the publisher throttling feature, controlled by the **ServerMaxMessages**, **TopicMaxMessages,** and **TopicMaxMessagesPad** settings.

Required Values

An integer between **0** and **99999**. The default value is **0**. If set to **0,** there is “no limit”; this causes the Server to create new files as needed, limited only by available disk space.

LockCacheIntoRAM

Description

The Windows VirtualLock API function locks the Server disk cache into physical memory, ensuring that subsequent access to the region will not incur a page fault (a swap-out to disk). This variable can be used in conjunction with **DBCacheSize** to improve performance.

Required Values

Yes or **No**. The default is **Yes**.

Note: *Only used in Windows, the administrator privilege is required.*

DBSync

Description

Specifies whether the Server controls the cache synchronization to disk:

- Keeping it set to the default, **True**, is the best guarantee of data integrity and Server reliability; for example, GEOD (Guaranteed Exactly Once Delivery) requires this parameter to be set to **True.**
- Setting it to False allows the operating system to control the synchronization schedule. Thus, if the operating system lazily flushes its disk write cache, the committed data may not actually be written to the disk at the time the server executes a **commit** operation.

Required Values

True or **False**. True specifies Message Service, while False specifies the operating system.

Message Settings

The **Message Settings** parameters govern message data memory and expiration settings on the SeeBeyond JMS IQ Manager (Server).

MaxPayloadMemory

Description

Specifies the maximum amount (in kilobytes) of message data payloads allowed to be in physical memory at any moment while the Server is running. When message data memory usage increases beyond the MaxPayloadMemory threshold, the Server will begin memory garbage collection and recovery.

Required Values

An integer between **1** and **999999999**. The default is **20000**. The upper limit depends on available memory resources.

PayloadMemoryPad

Description

When **MaxPayloadMemory** (see above—the threshold beyond which the server begins memory recovery and cleanup) is exceeded, the Server attempts to recover the exceeded memory plus a smallish amount more; the extra amount (in KB) is specified by the parameter **PayloadMemoryPad**.

Required Values

An integer between **0** and **999999**. The default is **100**. The upper limit must be less than **MaxPayloadMemory**.

MaxTimeToLive

Description

Specifies the maximum amount of time (in seconds) before a message expires. After it expires, the message is permanently scratched.

Required Values

An integer between **0** and **999999999**. The default is **2592000** (in other words, 30*24*60*60 seconds = 30 days). The special value **0** means “never expires”.

EnableEdit

Description

Turns on/off the ability to edit message contents.

Required Values

Yes or **No**.

EnableView

Description

Turns on/off the ability to view message contents.

Required Values

Yes or **No**.

EnableDelete

Description

Turns on/off the ability to delete messages.

Required Values

Yes or **No**.

Server Settings

Note: *The JMS term "topic" is used interchangeably with the e*Gate term "Event Type"; the JMS term "message" is used interchangeably with the e*Gate term "Event"; and the term "Server" is used generically for "SeeBeyond JMS IQ Manager."*

The **Server Settings** parameter sets the upper limit on the total number of messages that the Server will track before throttling publishers.

ServerMaxMessages

Description

Specifies the maximum total number of messages allowed in the server message queues. Used in conjunction with **STCMS.Topic.MaxMessages**. When the number of messages in the server reaches this specified value, the server starts throttling publishers based on the number of messages in their respective topic queues and the value of **STCMS.Topic.MaxMessages**. For more information, see the *SeeBeyond JMS Intelligent Queue User's Guide*.

Required Values

An integer between **0** and **999999999**. The default is **100000**. When set to **0**, publishers are never throttled.

Topic Settings

The **Topic Settings** parameters set the upper limit on the total number of messages the Server handles and govern per-topic traffic thresholds.

TopicMaxMessages

Description

Specifies the maximum number of messages permitted for any particular topic. When the number of messages on a topic reaches this value, all publishers of the topic are throttled. Once a publisher is throttled, the server stops reading messages from it until the number of topics in the queue it publishes to has dropped to below the threshold of (**TopicMaxMessages – TopicMaxMessages Pad**).

Required Values

An integer between **0** and **999999999**. The default is **1000**. If set to **0**, the publishers are never throttled.

TopicMaxMessagesPad

Description

Used in conjunction with **TopicMaxMessages** parameter. Specifies the number of messages that must be dequeued before publishers to the topic are unthrottled.

Required Values

An integer between **0** and **99999999**. The default is **100**. The value must be set to less than that of **TopicMaxMessages**.

Trace Settings

The **Trace Settings** parameters govern trace and debug logging behaviors.

TraceToFile

Description

Specifies whether informational, warning, and error messages are written to the log file.

Required Values

Yes or **No**. The default is **Yes**.

TraceLevel

Description

Specifies the trace print level.

Note: *For maximum debugging, use the setting **0**. To increase performance, you should set this to **1** or greater in production.*

Required Values

An integer between **0** and **3**. See below.

0	Info (in addition to all three categories below)
1	Warn (in addition to both categories below)
2	Error (in addition to the category below)
3	Fatal (only)

TraceMemory

Description

Sets memory-level tracing on or off.

Required Values

off or **on**. The default is **off**.

TraceToStdout

Description

Specifies whether to (also) print debugging/trace information to standard output.

Required Values

Yes or **No**. The default is **No**.

TraceVerbose

Description

Specifies whether debugging/trace information will issue complete full-length messages.

Required Values

Yes or **No**. The default is **No**.

TraceTimestamp

Description

Specifies whether to print timestamps in the debugging/trace log file.

Required Values

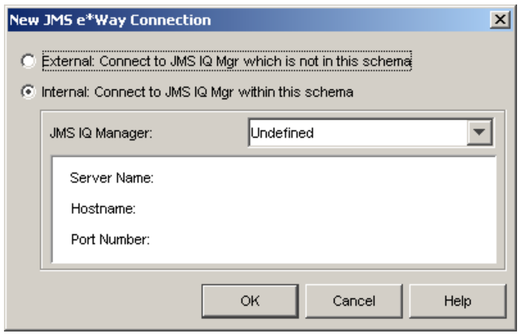
Yes or **No**. The default is **Yes**.

3.3 Configuring JMS e*Way Connection

The e*Way Connection provides the means for e*Gate to facilitate the exchange of data between the external system and e*Gate. The JMS e*Way Connection provides communication connectivity between e*Gate and the Message Server.

When you create the New e*Way JMS e*Way Connection configuration file, the following dialog box opens:

Figure 22 JMS e*Way Connection


New JMS e*Way Connection
External: Connect to JMS IQ Mgr which is not in this schema
Internal: Connect to JMS IQ Mgr within this schema
JMS IQ Manager:
Undefined
Server Name:
Hostname:
Port Number:
OK
Cancel
Help

Indicate whether the e*Way Connection is intended for:

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

If External is selected, the user must configure e*Way Connection, including ServerName, Hostname and Port Number.

If Internal is selected, the user selects a JMS IQ Manager from the drop-down, and the ServerName, Hostname and Port Number are read in from the Registry.

3.3.1 JMS e*Way Connection Parameters

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

This section describes the e*Way configuration parameters.

To change e*Way configuration parameters:

1. In the Enterprise Manager's Component editor, select the e*Way you want to configure and display its properties.
2. Under **Configuration File**, click **New** to create a new file, **Find** to select an existing configuration file, or **Edit** to edit the currently selected file.

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

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

- **General Settings** on page 46
- **Message Service** on page 47

General Settings

The General Settings control overall properties of the e*Way Connection.

Connection Type

Description

Specifies the type of connection to be established.

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

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

Required Values

Topic or **Queue**.

Transaction Type

Description

Specifies the type of transaction to be instantiated.

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

In **XA-compliant** (two-phase transactional style) a two-phase commit is done: The sender sends a prepare, and the commit occurs if and only if all receivers are prepared. Collaborations that use Guaranteed Exactly Once Delivery (GEOD) of Events require XA-compliant transaction types.

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

Required Values

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

Delivery Mode

Description

Setting **Delivery Mode** to **Persistent** guarantees that the JMS IQ Manager stores each message safely to disk. Setting it to **Non-Persistent** does not guarantee that the message is stored safely to disk. **Non-Persistent** provides better performance but no recovery.

Required Values

Non-Persistent or **Persistent**.

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

Maximum Number of Bytes to read

Description

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

Required Values

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

Default Outgoing Message Type

Description

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

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

Required Values

Bytes or **Text**.

Message Selector

Description

Specifies the Message Selector to be used for subscriptions.

Required Values

A string. The maximum length of query is set to 512 characters, including a null terminator. See **"The Message Selector" on page 76** for more information.

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

SeeBeyond Message Service Factory Class Name

Description

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

Required Values

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

Message Service

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

Server Name

Description

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

Required Values

A valid server name.

Host Name

Description

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

Required Values

A valid host name.

Port Number

Description

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

Required Values

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

Maximum Message Cache Size

Description

Specifies the maximum size of the message cache in bytes.

Required Values

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

Chapter 4

Implementing the Message Service

This chapter describes the implementation models, along with a sample implementation for the SeeBeyond Message Service.

4.1 Implementing Message Service Models

This section discusses how to use the JMS APIs and the JMS COM+ APIs to exchange data with an e*Gate system.

Considerations

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

1. The JMS Topic/Queue names and the e*Gate Event Types names must coincide.
2. The individual writing any external JMS code must know the expected data format, the name of the Topic/Queue, the name of host and port number of the JMS server.
3. The methods used must correspond to the expected data format.
4. For a list of e*Gate supported Java/COM+ classes, interfaces and methods, please see **“Client Libraries for the e*Gate Message Service” on page 115**.
5. The client code samples provided are intended to work directly with the sample schema provided. These are only samples created as a demonstration of possible behavior.

4.1.1 Message Overview

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

Message Structure

Message Service messages are composed of the following:

- Header - All messages support the same set of header fields. These header fields contain values that are used by both clients and providers to identify and route messages.
- Properties - Properties provide a means for adding optional header fields to messages
  - Application-specific
  - Standard properties
  - Provider-specific
- Body - JMS provides for supporting different types of message body contents, however, the current e*Way Connection supports bytes, and text messaging.

Message Header Fields

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

JMSDestination

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

JMSDeliverMode

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

JMSMessageID

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

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

JMSTimestamp

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

JMSExpiration

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

JMSRedelivered

The JMSRedelivered header filed contains the information that the message was re delivered to the consumer. If the header is “true”, the message is re delivered, and false if it’s not. The message may be marked as re delivered if a consumer fails to acknowledge delivery of the message, or if the JMS provider is uncertain that the consumer received the message.

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

JMSPriority

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

JMSReplyTo

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

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

JMSCorrelationID

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

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

JMSType

The JMSType header field is optionally set by the JMS client. The main purpose is to identify the message structure and the payload type. Not all vendors support this header.

Message Properties

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

Message Body

There are six types of message body or payload. Each form is defined by a message interface. Currently the following interfaces are supported by e*Gate:

- TextMessage - A message in which the body contains a java.lang.String. The inclusion of this message type is based on our presumption that String messages will be used extensively. It can be used to exchange both simple text messages and more complex data including XML documents.

- BytesMessage - A message in which a stream of uninterpreted bytes. This message type is for literally encoding a body to match an existing message format. It can be used for exchanging data in an application's native format or when JMS is being used purely as a transport between two systems.

4.2 Sample Code

The code samples that follow are available in the **samples** directory of the e*Gate CD-ROM, they are designed to work directly with the schema contained in the jmsdemo.zip in the directory:

samples\jmsapi

To download the samples, navigate to the following directory:

samples\jmsapi\com

samples\jmsapi\java

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

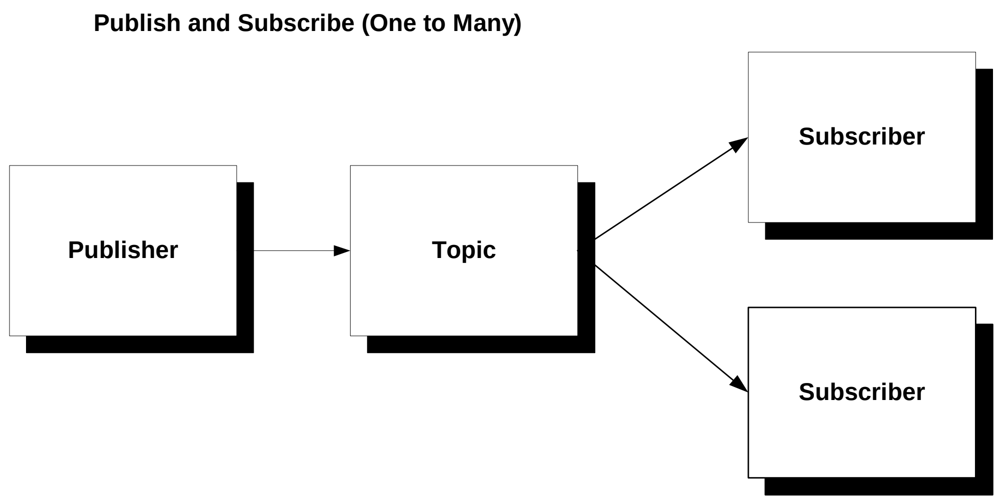
4.2.1 The Publish/Subscribe Model

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

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

Figure 23 below illustrates a basic Publish/Subscribe schema.

Figure 23 The Publish/Subscribe schema



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

Java Publish

The code sample below illustrates the following steps:

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

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

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

class Publisher {

    public static void main(String args[])
    {
        String hostName = "localhost";
        int port = 7555;
        try
        {
            System.out.println("-h(ost) host-name -p(ort) port-number");
            for (int i = 0; i < args.length; i++)
            {
                if (args[i].startsWith("-h") || args[i].startsWith("-host"))
                    hostName = args[++i];
                else if (args[i].startsWith("-p") || args[i].startsWith("-port"))
                    port = Integer.parseInt(args[++i]);
            }
        }
        catch(Exception e)
        {
            System.out.println("Error in arguments");
            System.exit(1);
        }
        String topicName = "PubSubSample";
        TopicConnectionFactory tcf = null;
        TopicConnection topicConnection = null;
        TopicSession topicSession = null;
        Topic topic = null;
        TopicPublisher topicPublisher = null;
        TextMessage message = null;
        final int MAX_MESSAGE_SIZE = 60;

        System.out.println("pub topic name is " + topicName);
        /*
         * Create connection.
         * Create session from connection; true means session is
         * transacted.
         * Create publisher and text message.
         * Send messages, varying text slightly.
         * Finally, close connection.
         */
        try {
            tcf = new STCTopicConnectionFactory(hostName, port);
            topicConnection = tcf.createTopicConnection();
            topicConnection.start();
            topicSession = topicConnection.createTopicSession(true,
            Session.AUTO_ACKNOWLEDGE);
            topic = topicSession.createTopic(topicName);
            topicPublisher = topicSession.createPublisher(topic);

            message = topicSession.createTextMessage();
            String s = new String("This is message.");
            message.setText(s);
            try {
                System.out.println("... Publishing message: " +
                    s);
                topicPublisher.publish(message);
                topicSession.commit();
            }
            catch (Exception exx) {
                exx.printStackTrace();
            }

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

Java Subscribe

The code sample below illustrates the following steps:

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

3 Create the subscriber.

4 Register message listener (TextListener).

5 When all messages have been received, enter “Q” to quit.

6 Close the connection.

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

```
import javax.jms.*;
import java.io.*;
import com.seebeyond.jms.client.*;
import com.seebeyond.jms.message.*;

public class Subscriber {

    public static void main(String args[]) {
        String hostName = "localhost";
        int port = 7555;
        try
        {
            System.out.println("-h(ost) host-name -p(ort) port-number");
            for (int i = 0; i < args.length; i++)
            {
                if (args[i].startsWith("-h") || args[i].startsWith("-host"))
                    hostName = args[++i];
                else if (args[i].startsWith("-p") || args[i].startsWith("-port"))
                    port = Integer.parseInt(args[++i]);
            }
        }
        catch(Exception e)
        {
            System.out.println("Error in arguments");
            System.exit(1);
        }
        String topicName = "eGatePubSubSample";
        TopicConnectionFactory tcf = null;
        TopicConnection topicConnection = null;
        TopicSession topicSession = null;
        Topic topic = null;
        TopicSubscriber topicSubscriber = null;
        STCBytesMessage message = null;
        InputStreamReader inputStreamReader = null;
        char answer = '\0';

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

        /*
         * Create connection.
         * Create session from connection; true means session is
         * transacted.
         * Create subscriber.
         * Register message listener (TextListener).
         * Receive text messages from topic.
         * When all messages have been received, enter Q to quit.
         * Close connection.
         */
        try {
            tcf = new STCTopicConnectionFactory(hostName, port);
            topicConnection = tcf.createTopicConnection();
            topicConnection.start();
            topicSession = topicConnection.createTopicSession(true,
            Session.AUTO_ACKNOWLEDGE);
            topic = topicSession.createTopic(topicName);
            topicSubscriber = topicSession.createSubscriber(topic);

            /*
             * Inner anonymous class that implements onMessage method
             * of MessageListener interface.
             *
```

```
                 */
                topicSubscriber.setMessageListener(new MessageListener(){
                    public void onMessage(Message message) {
                        STCTextMessage msg = null;
                        final int MAX_MESSAGE_SIZE = 60;
                        try {
                            if (message instanceof TextMessage) {
                                msg = (STCTextMessage) message;
                                System.out.println("... Reading message: " +
                                    msg.getText());
                            }
                            else {
                                System.out.println("Message of wrong type: " +
                                message.getClass().getName());
                            }
                        }
                        catch (Exception e) {
                            System.out.println("JMSException in onMessage(): "
                            + e.toString());
                        }
                        catch (Throwable te) {
                            System.out.println("Exception in onMessage():" +
                            te.getMessage());
                        }
                    }
                });
                topicSession.commit();
                System.out.println("To end program, enter Q or q, then <return>");
                inputStreamReader = new InputStreamReader(System.in);
                while (!((answer == 'q') || (answer == 'Q'))) {
                    try {
                        answer = (char)inputStreamReader.read();
                    }
                    catch (IOException e) {
                        System.out.println("I/O exception: " + e.toString());
                    }
                }
            }
            catch (JMSException e) {
                System.out.println("Exception occurred: " + e.toString());
                System.exit(1);
            }
            finally {
                if (topicConnection != null) {
                    try {
                        System.out.println("... Closing connection ...");
                        topicSession.commit();
                        topicConnection.close();
                    }
                    catch (JMSException e) {}
                }
            }
        }
    }
```

COM VB Publish/Subscribe

```
Option Explicit

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

Private Sub Form_Load()
```

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

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

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

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

ASP Publish

```
<%@ Language=VBScript %>


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

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

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

topicConnectionFactory.hostname = "JMS_Server_Machine"
```

```
'topicConnectionFactory.port = "JMS_Server_Port_Number"

Set topicConnection = topicConnectionFactory.CreateTopicConnection()

Set topicSession = topicConnection.CreateTopicSession(True,
AUTO_ACKNOWLEDGE)

topicConnection.Start

Set topic = topicSession.CreateTopic("test")

Set Publisher = topicSession.CreatePublisher(topic)

Set subscriber = topicSession.CreateSubscriber(topic)

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

Publisher.Publish MessagePublished

topicSession.Commit

Set MessageReceived = subscriber.Receive()

topicSession.Commit

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

%>

<P></P>

</BODY>
</HTML>
```

4.2.2 The Point-to-Point Model

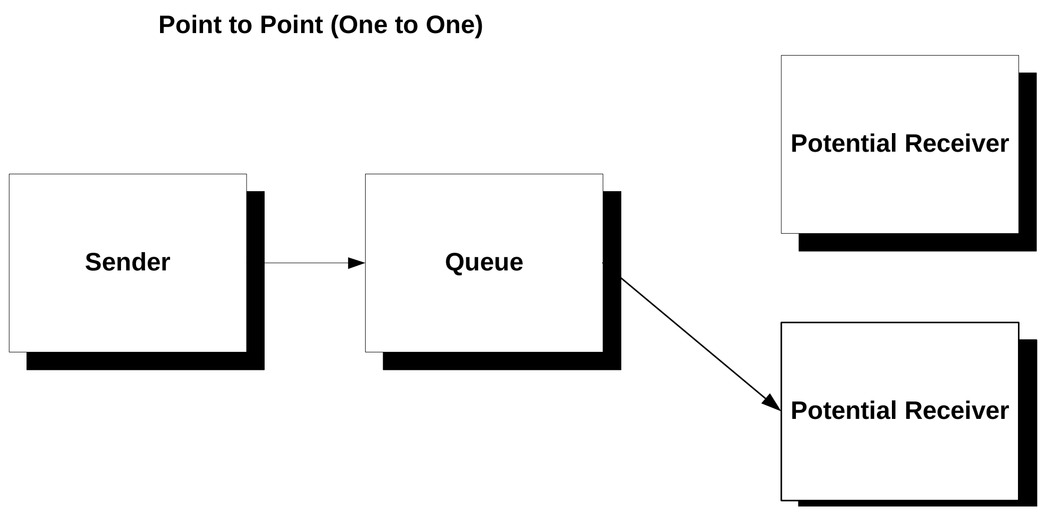
Point-to-Point messaging is based on the sending of a message to a named destination (as is the publish/subscribe model). There is no direct coupling of the producers to the consumers. One main difference between point-to-point and publish/subscribe messaging is that in the first, messages are delivered, without consideration of the current connection status of the receiver.

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

- Message exchange takes place via a queue. The queue acts as a destination to which producers send messages, and a source from which receivers consume messages.
- Each message is delivered to only one receiver. Multiple receivers may connect to a queue, but each message in the queue may only be consumed by one of the queue's receivers.
- The queue delivers messages to consumers in the order that they were placed in the queue by the message server. 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 24 Point to Point



Java Point-to-Point Sender

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


class Sender {

    public static void main(String args[]) {
        String hostName = "localhost";
        int port = 24053;
        try
        {
System.out.println("-h(ost) host-name -p(ort) port-number");
            for (int i = 0; i < args.length; i++)
            {
               if (args[i].startsWith("-h") || args[i].startsWith("-host"))
                    hostName = args[++i];
                else if (args[i].startsWith("-p") || args[i].startsWith("-
port"))
                    port = Integer.parseInt(args[++i]);
            }
        }
        catch(Exception e)
        {
            System.out.println("Error in arguments");
            System.exit(1);
        }

        String queueName = "P2PSample";
        QueueConnectionFactory qcf = null;
        QueueConnection queueConnection = null;
        QueueSession queueSession = null;
        Queue queue = null;
        QueueSender queueSender = null;
        TextMessage message = null;
        final int MAX_MESSAGE_SIZE = 60;
        System.out.println("pub queue name is " + queueName);
        /*
         * Create connection.
         * Create session from connection; true means session is
         * transacted.
         * Create sender and text message.
         * Send messages, varying text slightly.
         * Finally, close connection.
         */
        try {
            qcf = new STCQueueConnectionFactory(hostName, port);
            queueConnection = qcf.createQueueConnection();
            queueConnection.start();
            queueSession = queueConnection.createQueueSession(true,
                Session.AUTO_ACKNOWLEDGE);
            queue = queueSession.createQueue(queueName);
            queueSender = queueSession.createSender(queue);

            message = queueSession.createTextMessage();
            String s = new String("This is message ");
            message.setText(s);
            try {
                System.out.println("... Sending message: " +
                    s);
                queueSender.send(message);
                queueSession.commit();
            }
            catch (Exception exx) {
                exx.printStackTrace();
            }

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

Java Point-to-Point Receiver

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

public class Receiver {

    public static void main(String args[]) {
        String hostName = "localhost";
        int port = 24053;
        try
        {
            System.out.println("-h(ost) host-name -p(ort) port-number");
            for (int i = 0; i < args.length; i++)
            {
               if (args[i].startsWith("-h") || args[i].startsWith("-host"))
                    hostName = args[++i];
                else if (args[i].startsWith("-p") || args[i].startsWith("-
port"))
                    port = Integer.parseInt(args[++i]);
            }
        }
        catch(Exception e)
        {
            System.out.println("Error in arguments");
            System.exit(1);
        }
        String queueName = "eGateP2PSample";
        QueueConnectionFactory qcf = null;
        QueueConnection queueConnection = null;
        QueueSession queueSession = null;
        Queue queue = null;
        QueueReceiver queueReceiver = null;
        TextMessage message = null;
        InputStreamReader inputStreamReader = null;
        final Object syncObject = new Object();
        char answer = '\0';

        System.out.println("Queue name is " + queueName);

        /*
         * Create connection.
         * Create session from connection; true means session is
         * transacted.
         * Create receiver.
         * Register message listener (TextListener).
         * Receive text messages from queue.
         * When all messages have been received, enter Q to quit.
         * Close connection.
         */
        try {
            qcf = new STCQueueConnectionFactory(hostName, port);
            queueConnection = qcf.createQueueConnection();
            queueConnection.start();
            queueSession = queueConnection.createQueueSession(true,
                Session.AUTO_ACKNOWLEDGE);
            queue = queueSession.createQueue(queueName);
            queueReceiver = queueSession.createReceiver(queue);

            /*
             * Inner anonymous class that implements onMessage method
             * of MessageListener interface.
             *
             */
            queueReceiver.setMessageListener(new MessageListener(){
                public void onMessage(Message message) {
                    TextMessage msg = null;
                    final int MAX_MESSAGE_SIZE = 60;
                    try {
                        if (message instanceof TextMessage) {
                            msg = (TextMessage) message;
                            System.out.println("... Reading message: " +
                            msg.getText());
                        }
                        else {
                            System.out.println("Message of wrong type: " +
                            message.getClass().getName());
                        }
                        synchronized(syncObject)
                        {
                            syncObject.wait();
                        }
                    }
                    catch (InterruptedException ie)
                    {
                    }
                    catch (Exception e) {
                        System.out.println("JMSException in onMessage(): "
                        + e.toString());
                    }
                    catch (Throwable te) {
                        System.out.println("Exception in onMessage():" +
                        te.getMessage());
                    }
                }
            });
            queueSession.commit();
            System.out.println("-----To receive again, enter R or r, then
<return>");
            System.out.println("-----To end program, enter Q or q, then
<return>");
            inputStreamReader = new InputStreamReader(System.in);
            while (!((answer == 'q') || (answer == 'Q'))) {
                try {
                    answer = (char)inputStreamReader.read();
                    if (answer == 'r' || answer == 'R')
                    {
                        synchronized(syncObject)
                        {
                          syncObject.notifyAll();
                        }
                    }
                }
                catch (IOException e) {
                    System.out.println("I/O exception: " + e.toString());
                }
            }
        }
        catch (JMSException e) {
            System.out.println("Exception occurred: " + e.toString());
            System.exit(1);
        }
        finally {
            if (queueConnection != null) {
                try {
                    System.out.println("... Closing connection ...");
                    queueSession.commit();
                    queueConnection.close();
                }
                catch (JMSException e) {}
            }
        }
    }
}
```

COM VB Point-to-Point

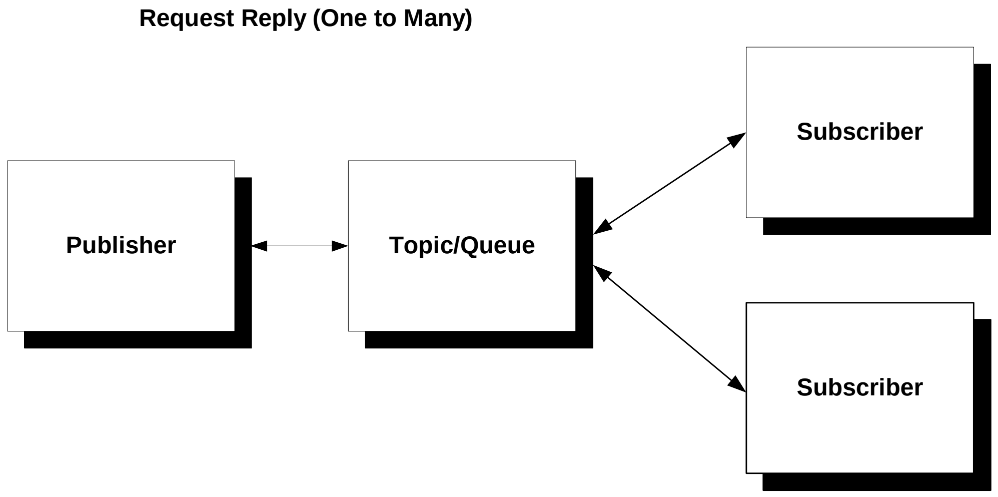
```
Option Explicit

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

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


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


End Sub



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

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

4.2.3 The Request-Reply Model

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

Figure 23 below illustrates a basic Request Reply schema.

Figure 25 The Request/Reply schema



1. A request is received by the **JMS Connection,** which is controlled by the JMS IQ Manager, the JMSReplyTo property is read into the internally directed by the Collaboration.
2. e*Gate reads in the request from **SampleTopicRequestor**, and appends a message to the end of the message for verification's sake.
3. The **SeeBeyond JMS IQ Manager** sends the message to a Temporary Topic via the JMS Connection.
4. The reply subscriber receives the message.
5. When the **Message Service** users disconnect, the temporary topic is destroyed.

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

Java Request/Reply

The code sample below illustrates the following steps:

1. Create the connection.
2. Create the session from connection (true indicates that the session is transacted).
3. Create the topic/queue and byte or text message.

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

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

Java TopicRequestor

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


class SampleTopicRequestor
implements ExceptionListener
{

/**
 * Main fuction create TopicRequestor and reply Subscriber.
 * Send reqeust message and wait for reply message.
 */

    public static void main(String args[])
    {

        SampleTopicRequestor listener = new SampleTopicRequestor();
        TopicConnectionFactory factory;
        TopicConnection requestConnection;
        String filename = null;
        File thisFile = null;
        String topicName = "TopicRequestorSample";
        String messageToSend = "SampleMessage";
        char[] myCharMessage = null;
        int fileLength = 0;
        BufferedReader stream = null;
        boolean output = true;
        String host = "localhost";
        int port = 24053;
        byte[] bytesFromEgate= new byte[64];
        String byteArrayStr = "";
        String usage = "Usage: java SampleTopicRequestor [-f/-m file/message] " +
                       "[-topic topic] [-host host] [-port port] " +
                       "[-help] [-output true/false]";
             String help = usage + "\n\n" +
                       "-f <file name>\n" +
                          "-m <message: default SampleMessage>\n" +
                       "-topic <topic name: default TopicRequestor>\n" +
                       "-host <host name where ms server is running: default
localhost>\n" +
                       "-port <port where ms server is running: default 24053>\n" +
                       "-output <display output message or not: default true>\n" +
                         "-help <this screen>\n";


        for(int i = 0; i < args.length; i++) {
            if( args[i].equals("-f") ) {
               filename = args[++i];
               thisFile = new File(filename);
               if( thisFile.canRead() ) {
                   try {
                       fileLength = (int) thisFile.length();
                       myCharMessage = new char[fileLength + 1];
                       stream = new BufferedReader(new InputStreamReader(new
FileInputStream(filename)));
                       stream.read(myCharMessage, 0, fileLength);
                       messageToSend = new String( myCharMessage );
                   }
                   catch( IOException e) {
                      e.printStackTrace();
                   }
               }
               }
               else if( args[i].equals("-m") )
                  messageToSend = args[++i];
               else if ( args[i].equals("-topic") )
                  topicName = args[++i];
               else if ( args[i].equals("-host") )
                  host = args[++i];
               else if( args[i].equals("-port") )
                  port = Integer.parseInt(args[++i]);
               else if( args[i].equals("-output") )
                   output = Boolean.getBoolean(args[++i]);
               else if( args[i].equals("-help") ) {
                   System.out.println(help);
                   System.exit(0);
               }
               else {
                   System.out.println(usage);
                   System.exit(1);
               }
           }

           try
           {
                // Create TopicConnection
                factory = new STCTopicConnectionFactory(host,port);
                requestConnection = factory.createTopicConnection();
                requestConnection.start();

                // Set the ExceptionListener
                requestConnection.setExceptionListener(listener);

                // Create TopicSession
               TopicSession topicSession = requestConnection.createTopicSession
(false,
                    Session.AUTO_ACKNOWLEDGE);
                // Create Topic
                Topic topic = topicSession.createTopic(topicName);
                // Create TopicRequestor
                STCTopicRequestor requestor = new STCTopicRequestor
(topicSession,topic);

                // Create TextMessage
                TextMessage textMessage = topicSession.createTextMessage();
                textMessage.setText(messageToSend);
                TextMessage replyTextMessage = (TextMessage)
requestor.request(textMessage);
                if( output )
                  System.out.println("... Got message: " + replyTextMessage.getText());


               System.out.println("... SampleTopicRequestor finished.");
               requestConnection.close();
           }
           catch(JMSException je)
           {
               je.printStackTrace();
           }
       }

       public void onException(JMSException e)
       {
           e.printStackTrace();
       }
}
```

Java QueueRequestor

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


class SampleQueueRequestor
implements ExceptionListener
{

/**
 * Main fuction create QueueRequestor and reply Subscriber.
 * Send reqeust message and wait for reply message.
 */

    public static void main(String args[])
    {

        SampleQueueRequestor listener = new SampleQueueRequestor();
        QueueConnectionFactory factory;
        QueueConnection requestConnection;
        String filename = null;
        File thisFile = null;
        String queueName = "QueueRequestorSample";
        String messageToSend = "SampleMessage";
        char[] myCharMessage = null;
        int fileLength = 0;
        BufferedReader stream = null;
        boolean output = true;
        String host = "localhost";
        int port = 24053;
        byte[] bytesFromEgate= new byte[64];
        String byteArrayStr = "";
        String usage = "Usage: java SampleQueueRequestor [-f/-m file/
message] " +
                       "[-queue queue] [-host host] [-port port]" +
                       "[-help] [-output true/false]";
            String help = usage + "\n\n" +
                       "-f <file name>\n" +
                        "-m <message: default SampleMessage>\n" +
                       "-queue <queue name: default QueueRequestor>\n" +
                      "-host <host name where ms server is running: default
localhost>\n" +
                       "-port <port where ms server is running: default
24053>\n" +
                       "-output <display output message or not: default
true>\n" +
                       "-help <this screen>\n";


        for(int i = 0; i < args.length; i++) {
            if( args[i].equals("-f") ) {
               filename = args[++i];
               thisFile = new File(filename);
               if( thisFile.canRead() ) {
                   try {
                       fileLength = (int) thisFile.length();
                       myCharMessage = new char[fileLength + 1];
                      stream = new BufferedReader(new InputStreamReader(new
FileInputStream(filename)));
                       stream.read(myCharMessage, 0, fileLength);
                       messageToSend = new String( myCharMessage );
                   }
                   catch( IOException e) {
                      e.printStackTrace();
```

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

        try
        {
            // Create QueueConnection
            factory = new STCQueueConnectionFactory(host,port);
            requestConnection = factory.createQueueConnection();
            requestConnection.start();

            // Set the ExceptionListener
            requestConnection.setExceptionListener(listener);

            // Create QueueSession
           QueueSession queueSession = requestConnection.createQueueSession
(false,
                Session.AUTO_ACKNOWLEDGE);
            // Create Queue
            Queue queue = queueSession.createQueue(queueName);
            // Create QueueRequestor
            STCQueueRequestor requestor = new STCQueueRequestor
(queueSession,queue);

            // Create TextMessage
            TextMessage textMessage = queueSession.createTextMessage();
            textMessage.setText(messageToSend);
            TextMessage replyTextMessage = (TextMessage)
requestor.request(textMessage);
            if( output )
                System.out.println("... Got message: " +
replyTextMessage.getText());


            System.out.println("... SampleQueueRequestor finished.");
            requestConnection.close();
        }
        catch(JMSException je)
        {
            je.printStackTrace();
        }
    }

    public void onException(JMSException e)
    {
        e.printStackTrace();
        System.exit(1);
    }
}
```

COM VB TopicRequestor

```
Option Explicit

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

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

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

COM VB QueueRequestor

```
Option Explicit

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

Private Sub Form_Load()
' You should replace the host name and port number with the actual 
' values
    queueConnectionFactory.HostName = "localhost"
    queueConnectionFactory.Port = 24053
' Create a queue connection
```

```
    Set queueConnection =
queueConnectionFactory.CreateQueueConnection()
' Create a queue session
    Set queueSession = queueConnection.CreateQueueSession(False,
msAutoAcknowledge)
' Start the session
    queueConnection.Start
End Sub

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

4.2.4 JNDI

To use JNDI in your programs, you need to set up its compilation and execution environments. The following are the JNDI packages:

- `javax.naming`: Provides the classes and interfaces for accessing naming services.
  - `Context`: Represents a naming context, which consists of a set of name-to-object bindings.
  - `Name`: Represents a generic name for an ordered sequence of components.
  - `NameParser`: Used for parsing names from a hierarchical namespace.
  - `NamingEnumeration`: Used for enumerating lists returned by methods in the `javax.naming` and `javax.naming.directory` packages.
  - `Referenceable`: Implemented by an object that provides a Reference to itself.
    - `BinaryRefAddr`: Binary form of the address for a communications end-point
    - `Binding`: Name/Object binding found as found in a context.
    - `CompositeName`: Composite name, as a sequence of names spanning multiple name spaces.
    - `CompoundName`: Compound name, as a name contained in a name space.
    - `InitialContext`: Starting context for performing naming operations.
    - `LinkRef`: A reference whose contents is a link name, which is bound to the atomic name in a context.
    - `NameClassPair`: Object name and class name pair as found in a context.
    - `RefAddr`: Address of a communications end-point.

    - `Reference`: Reference to an object as found in the naming/directory system.
    - `StringRefAddr`: String for of the address for a communications end-point.

- `javax.naming.directory`: Extends the `javax.naming` package to provide functionality for accessing directory services.
  - `Attribute`:Represents an attribute associated with a named object.
  - `Attributes`: Represents a collection of attributes.
  - `DirContext`: Directory service interface, contains the methods for examining and updating attributes associated with objects, and for searching the directory.
    - `BasicAttribute`: Class provides basic implementation of `Attribute` interface.
    - `BasicAttributes`: Class provides a basic implementation of the `Attributes` interface.
    - `InitialDirContext`: Class is the starting context for performing directory operations.
    - `ModificationItem`: Class represents a modification item.
    - `SearchControls`: Class encapsulates factors that determine the scope of the search, along with the returns from that search.
    - `SearchResult`: Class represents an item in the `NamingEnumeration` returned as a result of the `DirContext.search()` methods.
- `javax.naming.event:` Provides support for event notification when accessing naming and directory services.
  - `EventContext`: Contains the methods for registering/de registering listeners to be notified of events fired, when objects named in a context change.
  - `EventDirContext`: Contains methods for registering listeners to be notified of events fired when objects named in a directory context change.
  - `NamespaceChangeListener`: The root of listener interfaces that handle `NamingEvent`s.
  - `NamingListener`: The root of listener interfaces that handle `NamingEvents`.
  - `ObjectChangeListener`: Specifies the method that a listener of a `NamingEvent` with event type of `OBJECT_CHANGED` must implement.
    - `NamingEvent`: Class represents an event fired by a naming/directory service.
    - `NamingExceptionEvent`: Class represents an event fired when the procedures/processes are used to collect information from listeners of `NamingEvents` that throw a `NamingException`.
- `javax.naming.ldap:`Provides support for LDAPv3 extended operations and controls.
  - `Control`: Represents an LDAPv3 control as defined in **RFC2251**

- `ExtendedRequest`: Represents an LDAPv3 extended operation request as defined in **RFC2251**
- `HasControls`: Used for returning controls with objects returned in NamingEnumerations.
- `LdapContext`: Represents a context in which you can perform operations with LDAPv3-style controls and perform LDAPv3-style extended operations.
- `UnsolicitedNotification`: Represents an unsolicited notification as defined in **RFC2251**
- `UnsolicitedNotificationListener`: Handles `UnsolicitedNotificationEvent`.
  - `ControlFactory`: An abstract class that represents a factory for creating LDAPv3 controls.
  - `InitialLdapContext`: The class is the starting context for performing LDAPv3 style extended operations and controls.
  - `StartTlsRequest`: The class implements the LDAPv3 Extended Request for STartTLS as defined in *Lightweight Directory Access Protocol (v3): Extension for Transport Layer Security*.
  - `StartTlsResponse`: The class implements the LDAPv3 Extended Response for StartTLS as defined in *Lightweight Directory Access Protocol (v3): Extension for Transport Layer Security.*
  - `UnsolicitedNotificationEvent`: This class represents an event fired in response to an unsolicited notification sent by the LDAP server.
- `javax.naming.spi:`Provides the means for dynamic plug-in to naming and directory services via `javax.naming` and related packages.

To compile programs that use JNDI, access to JNDI classes are required. If you are using Java 2 SDK v1.3 or higher, the JNDI classes are already included, and no further action is required. If you are using an older version of the Java SDK, then you need to download the JNDI classes from the JNDI Web site (**http://java.sun.com/products/jndi/**).

At runtime, you will also require access to the classes for any service providers that your program uses. The Java 2 Runtime Environment (JRE) v1.3 already includes the JNDI classes and service providers for LDAP, COS naming, and the RMI registry. If you are using some other service Providers, then you need to download and install the associated archive files in the classpath, JAVA_HOME/jre/lib/ext directory, where JAVA_HOME is the directory containing the JRE (**http://java.sun.com/j2se/1.3/**).

If you are not using JNDI as an installed extension or are using the JRE v1.1, then copy the JNDI and service provider archive files to their permanent location and add that location to your classpath. You can do that by a setting the `CLASSPATH` variable to include the absolute filenames of the archive files.

Initial Context

Before performing any operation on a naming or directory service, an initial context must be acquired, providing a starting point into the name space. All methods used to access naming and directory services are performed relative to some context. To obtain an initial context, perform the following:

1 Select the service provider of the corresponding service to which to access.

   Specify the service provider to use for the initial context by creating a set of environment properties (a `Hash table`) and adding the name of the service provider class to it.

2 Specify any configuration required by the initial context.

   Different clients might require various information for contacting the desired directory. For example, the machine upon which the server is running, and the user identity, might be required. This information is passed to the service provider via environment properties. Consult your service provider documentation for more information.

3 Call the `InitalContext` constructor (**http://java.sun.com/j2se/1.3/docs/api/javax/naming/InitialContext.html#constructor_detail**).

   The environment properties, previously created are passed to the `InitialContext` constructor. A reference to a **Context** object is now available to access the naming service. To perform directory operations, **InitialDirContext** is used.

Naming Operations

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

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

Looking Up an Object

In the following JNDI sample code, the naming operation `lookup().` The name of the object to be "looked up" is passed in, relative to the current context, and the object is retrieved. JMS provider implementations of administered objects should be both `javax.jndi.Referenceable` and `java.io.Serializable` so that they can be stored in all JNDI naming contexts.

JNDI Samples

This sample requires edits be made to generic information. See the code samples below in bold type-set.

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

C:\temp>set
CLASSPATH=%CLASSPATH%;C:\jndi\fscontext1_2beta3\lib\fscontext.jar;C:\jndi\f
scontext1_2beta3\lib\providerutil.jar

Each of the following samples requires the following imports:

```
import javax.jms.*;
import javax.naming.*;
import com.seebeyond.jms.client.*;
import java.util.Properties;
```

A sample class is declared:

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

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

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

QueueConnectionFactory Sample

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

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

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

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

Queue Sample

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

```
// Instantiate a SeeBeyond Queue object and bind to JNDI
        STCQueue que = new STCQueue("AccountsPayableQueue");
        try {
        jndi.bind("APQueue", (Queue) que);
        }
        catch (javax.naming.NameAlreadyBoundException e) {
        }

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

TopicConnectionFactory Sample

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

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

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

Topic Sample

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

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

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

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

At this point the retrieved objects can now communicate with the SeeBeyond Message Service and e*Gate.

4.2.5 The Message Selector

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

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

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

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

Message Selector Syntax

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

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

Identifiers

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

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

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

The following JMS headers can be used as identifiers:

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

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

- JMSDestination
- JMSReplyTo

The JMSRedelivered value may be changed during delivery. For example, if a consumer users a message selector where "JMSRedelivered = FALSE", and there was a failure delivering a message, the JMSRedelivered flag might be set to TRUE. JMSExpiration is not supported as an identifier, because JMS providers may choose to implement this value in different manners. (Some may store it with the message, while others calculate it as needed.)

Literals

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

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

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

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

Declaring a Message Selector

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

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

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

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

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

Java Message Selector Publisher

```
import javax.jms.*;
import com.seebeyond.jms.client.STCTopicConnectionFactory;
import com.seebeyond.jms.util.*;

public class SelectorPublisher
{

    private static final String HOST = "localhost";
```

```
        private static final int PORT = 24053;
        private static final String PROP_NAME = "property";
        private static final String TOPIC_NAME = "Selector";

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

            try
            {
                TopicConnectionFactory factory = new
   STCTopicConnectionFactory(hostName, port);
                TopicConnection conn = factory.createTopicConnection();
                conn.start();
                TopicSession topicSession = conn.createTopicSession(true,
   Session.AUTO_ACKNOWLEDGE);

                // create temporary queue
                Topic topic = topicSession.createTopic(TOPIC_NAME);

                // create sender
                TopicPublisher publisher = topicSession.createPublisher(topic);
                publisher.setDeliveryMode(DeliveryMode.PERSISTENT);

                // put messages on queue
                for(int ii=0; ii<10 ;ii++)
                {
                    TextMessage msg = topicSession.createTextMessage();
                    int index = ii%10;
                    msg.setStringProperty(PROP_NAME, ""+index);
                    msg.setText("This is message body.");
                    publisher.publish(msg);
                    System.out.println("... Published 1 message with
   "+PROP_NAME+" = "+ii);
                }
                topicSession.commit();
                conn.close();

            } catch(Exception ex)
            {
                ex.printStackTrace();
                System.exit(1);
            }
        }
```

```
}
```

Java Message Selector Subscriber

```
import javax.jms.*;
import com.seebeyond.jms.client.STCTopicConnectionFactory;
import com.seebeyond.jms.util.*;

public class SelectorSubscriber
{

    private static final String HOST = "localhost";
    private static final int PORT = 24053;
    private static final String PROP_NAME = "property";
    private static final String TOPIC_NAME = "eGateSelector";

    public static void main(String[] args)
    throws Exception
    {
              String hostName = HOST;
              int port = PORT;
        int selector = 7;
        try
        {
            System.out.println("-h(ost) host-name -p(ort) port-number -
s(elector) property-value-from-0-to-9");
            for (int i = 0; i < args.length; i++)
            {
               if (args[i].startsWith("-h") || args[i].startsWith("-host"))
                    hostName = args[++i];
                else if (args[i].startsWith("-p") || args[i].startsWith("-
port"))
                    port = Integer.parseInt(args[++i]);
                else if (args[i].startsWith("-s") || args[i].startsWith("-
selector"))
                    selector = Integer.parseInt(args[++i]);
            }
        }
        catch(Exception e)
        {
            System.out.println("Error in arguments");
            System.exit(1);
        }
        try
        {
            TopicConnectionFactory factory = new
STCTopicConnectionFactory(hostName, port);
            TopicConnection conn = factory.createTopicConnection();
            conn.start();
            TopicSession topicSession = conn.createTopicSession(true,
Session.AUTO_ACKNOWLEDGE);

            // create temporary queue
            Topic topic = topicSession.createTopic(TOPIC_NAME);

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

```
            // receive message
            for (Message msg = subscriber.receive();
                msg != null;
                msg = subscriber.receive(1000))
            {
               System.out.println("... Received 1 message with "+ PROP_NAME
+ " = " + msg.getStringProperty(PROP_NAME));
            }
            topicSession.commit();
            subscriber.close();
            topicSession.unsubscribe("SelectorSubscriber"+selector);

            conn.close();
        }
        catch(Exception ex)
        {
            ex.printStackTrace();
            System.exit(1);
        }
    }


}
```

COM VB Message Selector

```
Option Explicit

Dim MessageSelector As String
Dim topicConnectionFactory As New topicConnectionFactory
Dim topicConnection As topicConnection
Dim topicSession As topicSession
Dim topic As topic
Dim Publisher As TopicPublisher
Dim subscriber As TopicSubscriber
Dim MessagePublished  As TextMessage
Dim MessageReceived As TextMessage

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

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

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

End Sub
```

4.2.6 XA Sample

XA compliance is achieved when cooperating software systems contain sufficient logic to ensure that the transfer of a single unit of data between those systems is neither lost nor duplicated because of a failure condition in one or more of the cooperating systems. e*Gate 4.5 and later satisfies this requirement via utilization of the XA Protocol, from the X/Open Consortium.

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

Java XA Publisher

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

class XAPublisher {

    public static void main(String args[]) {
        String hostName = "localhost";
        int port = 24053;
        try
        {
            System.out.println("-h(ost) host-name -p(ort) port-number");
            for (int i = 0; i < args.length; i++)
            {
                if (args[i].startsWith("-h") || args[i].startsWith("-host"))
                    hostName = args[++i];
                 else if (args[i].startsWith("-p") || args[i].startsWith("-
port"))
                    port = Integer.parseInt(args[++i]);
            }
        }
        catch(Exception e)
        {
            System.out.println("Error in arguments");
            System.exit(1);
        }
        String topicName = "XAPubSubSample";
        XATopicConnectionFactory tcf = null;
        XATopicConnection topicConnection = null;
        XATopicSession xaTopicSession = null;
        XAResource resource = null;
        TopicSession topicSession = null;
        Topic topic = null;
        TopicPublisher topicPublisher = null;
        TextMessage message = null;
        final int MAX_MESSAGE_SIZE = 60;
        InputStreamReader inputStreamReader = null;
        char answer = '\0';

        System.out.println("pub topic name is " + topicName);
        /*
         * Create connection.
         * Create session from connection; true means session is
         * transacted.
         * Create publisher and text message.
         * Send messages, varying text slightly.
         * Finally, close connection.
         */
        try {
            tcf = new STCXATopicConnectionFactory(hostName, port);
            topicConnection = tcf.createXATopicConnection();
            topicConnection.start();
            xaTopicSession = topicConnection.createXATopicSession();
            resource = xaTopicSession.getXAResource();
            topicSession = xaTopicSession.getTopicSession();
            topic = topicSession.createTopic(topicName);
            topicPublisher = topicSession.createPublisher(topic);
            Xid xid = new XidImpl();

            byte[] mydata = new byte[MAX_MESSAGE_SIZE];

            message = topicSession.createTextMessage();
            String s = new String("This is message ");
            message.setText(s);
            inputStreamReader = new InputStreamReader(System.in);
            try {
                System.out.println("... XAResource start");
                resource.start(xid, XAResource.TMNOFLAGS);
                System.out.println("... Publishing message: " +
                    s);
                topicPublisher.publish(message);
                System.out.println("... XAResource end");
                resource.end(xid, XAResource.TMSUCCESS);
                System.out.println("XAResource prepare ....");
                resource.prepare(xid);
                System.out.println("C or c to commit and R or r to
rollback");
                while (true)
                {
                    answer = (char) inputStreamReader.read();
                    if (answer == 'c' || answer == 'C')
                    {
                        System.out.println("... XAResource commit");
                        resource.commit(xid, false);
                        break;
                    }
                    else if (answer == 'r' || answer == 'R')
                    {
                        System.out.println("... XAResource rollback");
                        resource.rollback(xid);
                        break;
                    }
                }
            }
            catch (Exception exx) {
                exx.printStackTrace();
            }

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

Java XA Subscriber

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

public class XASubscriber {

    public static void main(String args[]) {
        String hostName = "localhost";
        int port = 24053;
        try
        {
            System.out.println("-h(ost) host-name -p(ort) port-number");
            for (int i = 0; i < args.length; i++)
            {
               if (args[i].startsWith("-h") || args[i].startsWith("-host"))
                    hostName = args[++i];
                else if (args[i].startsWith("-p") || args[i].startsWith("-
port"))
                    port = Integer.parseInt(args[++i]);
            }
        }
        catch(Exception e)
        {
            System.out.println("Error in arguments");
            System.exit(1);
```

```
}
String topicName = "eGateXAPubSubSample";
XATopicConnectionFactory tcf = null;
XATopicConnection topicConnection = null;
XATopicSession xaTopicSession = null;
XAResource resource = null;
TopicSession topicSession = null;
Topic topic = null;
TopicSubscriber topicSubscriber = null;
TextMessage message = null;
Xid xid = null;
InputStreamReader inputStreamReader = null;

char answer = '\0';

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

/*
 * Create connection.
 * Create session from connection; true means session is
 * transacted.
 * Create subscriber.
 * Register message listener (TextListener).
 * Receive text messages from topic.
 * When all messages have been received, enter Q to quit.
 * Close connection.
 */
try {
    tcf = new STCXATopicConnectionFactory(hostName, port);
    topicConnection = tcf.createXATopicConnection();
    topicConnection.start();
    xaTopicSession = topicConnection.createXATopicSession();
    resource = xaTopicSession.getXAResource();
    topicSession = xaTopicSession.getTopicSession();
    topic = topicSession.createTopic(topicName);
    xid = new XidImpl();
    System.out.println("... XAResource start");
    resource.start(xid, XAResource.TMNOFLAGS);
    topicSubscriber = topicSession.createSubscriber(topic);

    /*
     * Inner anonymous class that implements onMessage method
     * of MessageListener interface.
     *
     */
    topicSubscriber.setMessageListener(new MessageListener(){
        public void onMessage(Message message) {
            TextMessage msg = null;
            try {
                if (message instanceof TextMessage) {
                    msg = (TextMessage) message;
                    System.out.println("... Reading message: " +
                        msg.getText());
                }
                else {
                    System.out.println("Message of wrong type: " +
                    message.getClass().getName());
                }
            }
            catch (Exception e) {
                System.out.println("JMSException in onMessage(): "
                + e.toString());
            }
            catch (Throwable te) {
```

```
                        System.out.println("Exception in onMessage():" +
                        te.getMessage());
                    }
                }
            });

            BufferedReader reader = new BufferedReader(new
InputStreamReader(System.in));
            while (!((answer == 'q') || (answer == 'Q'))) {
                try {
                    System.out.println("C or c to commit and R or r to
rollback after prepare");
                    answer = (char)reader.readLine().charAt(0);
                    System.out.println("... XAResource end");
                    resource.end(xid, XAResource.TMSUCCESS);
                    System.out.println("... XAResource prepare");
                    resource.prepare(xid);
                    if (answer == 'c' || answer == 'C')
                    {
                        System.out.println("... XAResource commit");
                        resource.commit(xid, false);
                    }
                    else
                    {
                        System.out.println("... XAResource rollback");
                        resource.rollback(xid);
                    }
                    System.out.println("... XAResource start");
                resource.start(xid, XAResource.TMNOFLAGS);
                    System.out.println("To end program, enter Q or q, then
<return>. To continue receive, enter r or R");
                    answer = (char)reader.readLine().charAt(0);
                }
                catch (IOException e) {
                    System.out.println("I/O exception: " + e.toString());
                }
            }

        }
        catch (Exception e) {
            System.out.println("Exception occurred: " + e.toString());
            System.exit(1);
        }
        finally {
           if (topicConnection != null && resource != null && xid != null) {
                try {
                    System.out.println("... XAResource end");
                    resource.end(xid, XAResource.TMSUCCESS);
                    System.out.println("... XAResource prepare");
                    resource.prepare(xid);
                    System.out.println("... XAResource commit");
                    resource.commit(xid, false);
                    System.out.println("... Closing connection");
                    topicConnection.close();
                }
                catch (Exception e) {}
            }
        }
    }
}
```

COM VB XA Sample

```
Option Explicit
```

```
Dim TopicObj As TopicTask
Dim QueueObj As QueueTask

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

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

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

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

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

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

4.2.7 The Compensating Resource Manager

Creating an SQL Database

The samples provided are designed using an SQL Database.

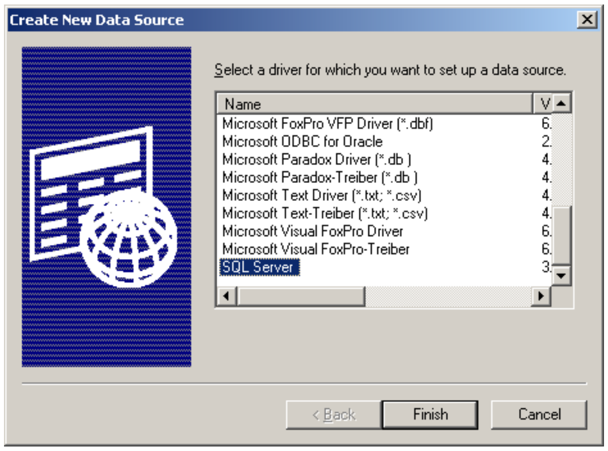
1 Create an SQL database, using the name “CRM” for the purpose of testing the samples.

2 Create a table, using the name “Messages”.

3 Create two columns in the table, “UID” and “Message”.

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

Figure 26 SQL Database Source


Create New Data Source

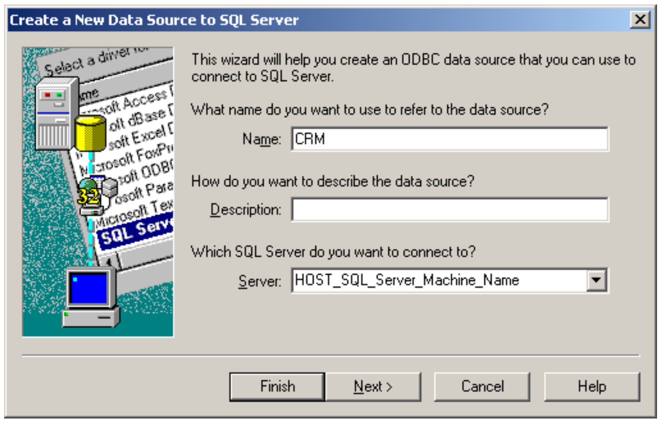
Select a driver for which you want to set up a data source.

Name	V
Microsoft FoxPro VFP Driver (*.dbf)	6.
Microsoft ODBC for Oracle	2.
Microsoft Paradox Driver (*.db)	4.
Microsoft Paradox-Treiber (*.db)	4.
Microsoft Text Driver (*.txt; *.csv)	4.
Microsoft Text-Treiber (*.txt; *.csv)	4.
Microsoft Visual FoxPro Driver	6.
Microsoft Visual FoxPro-Treiber	6.
SQL Server	3.

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

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

Figure 27 SQL Datasource


Create a New Data Source to SQL Server

This wizard will help you create an ODBC data source that you can use to connect to SQL Server.

What name do you want to use to refer to the data source?

Name: CRM

How do you want to describe the data source?

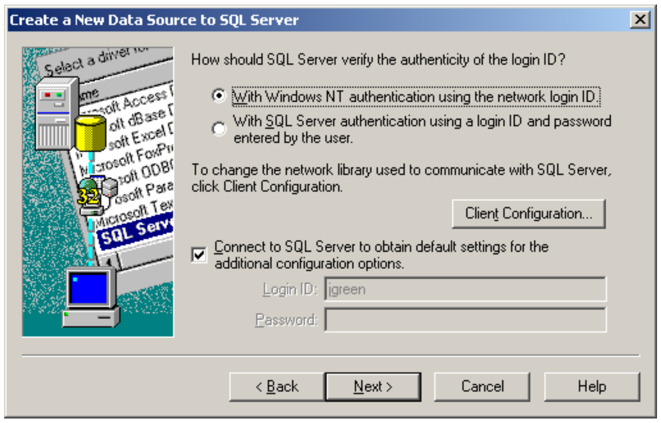
Description:

Which SQL Server do you want to connect to?

Server: HOST_SQL_Server_Machine_Name

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

Figure 28 Login Settings



How should SQL Server verify the authenticity of the login ID?

[✓ With Windows NT authentication using the network login ID.]

With SQL Server authentication using a login ID and password entered by the user.

To change the network library used to communicate with SQL Server, click Client Configuration.

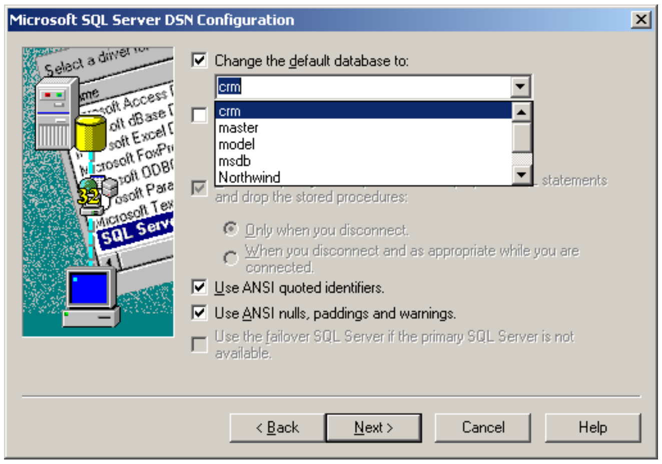
[✓] Connect to SQL Server to obtain default settings for the additional configuration options.

Login ID: jgreen

Password:

7 Select the recently created database as the default from the drop-down as in the figure below. Click Next to Continue.

Figure 29 Default SQL Database



[✓] Change the default database to: crm

[✓] ... statements and drop the stored procedures:

[✓ Only when you disconnect.]

When you disconnect and as appropriate while you are connected.

[✓] Use ANSI quoted identifiers.

[✓] Use ANSI nulls, paddings and warnings.

Use the failover SQL Server if the primary SQL Server is not available.

8 Click Finish.

Configuring the Compensating Resource Manager (CRM)

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

Both the CRM Worker and Compensator are COM+ components and they must be configured using the Windows 2000 Component Services Explorer function properly. The CRM Worker and CRM Compensator must be installed into the same COM+ application that was completed above. (See **"Configuring the Compensating Resource Manager (CRM)" on page 88** for more information.)

Note: *You must create the "CRM" database before attempting to use any sample code. See* **"Creating an SQL Database" on page 86** *for more information.*

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

CRMTest.vdb

CRMTest.dll

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

InsertMessage.cls

```
Option Explicit

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

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

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

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

TwoTasks.cls

```
Option Explicit
' In this CRM sample, there are two tasks that must either both succeed
' or both abort
' The TopicTask is to send a message to the JMS server
Dim topicObj As TopicTask
' The InsertMessage task is to insert a message into a SQL table
Dim MessageObj As InsertMessage

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

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

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

TopicTask.cls

```
Option Explicit

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

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

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

    Sub Receive(topicName As String, msg As String, commit As Boolean)
        On Error GoTo errHandler
        Dim ObjCtx As ObjectContext
        Set ObjCtx = GetObjectContext()
        If ObjCtx Is Nothing Then
            MsgBox "Application is not running in COM+"
            Exit Sub
        End If
    ' Create a XA topic connection factory
        Set XATopicConnectionFactory = New XATopicConnectionFactory
    ' You should replace the host name and port number with the actual values
        XATopicConnectionFactory.HostName = "localhost"
        XATopicConnectionFactory.Port = 24053
    ' Create a XA topic connection
        Set XATopicConnection =
    XATopicConnectionFactory.CreateXATopicConnection()
    ' Create a XA topic session
        Set XATopicSession = XATopicConnection.CreateXATopicSession()
        Set TopicSession = XATopicSession.TopicSession
    ' Create a topic
        Set Topic = TopicSession.CreateTopic(topicName)
    ' Start the XA topic session
        XATopicConnection.Start
    ' Create a subscriber
        Set subscriber = TopicSession.CreateDurableSubscriber(Topic, 
    "TopicSubscriber")
    ' receive a message
        Set MessageReceived = subscriber.ReceiveNoWait
        If MessageReceived Is Nothing Then
            msg = "No Message Received"
        Else
            msg = MessageReceived.Text
        End If
        If Not commit Then
            ObjCtx.SetAbort
            Err.Raise vbObjectError + 1024, , "Instruction to Abort the 
    Transaction"
        End If
        Exit Sub
    errHandler:
        ObjCtx.SetAbort
        Err.Raise Err
    End Sub
```

QueueTasks.cls

```
Option Explicit

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

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

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

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

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

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

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

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

4 From the following location:

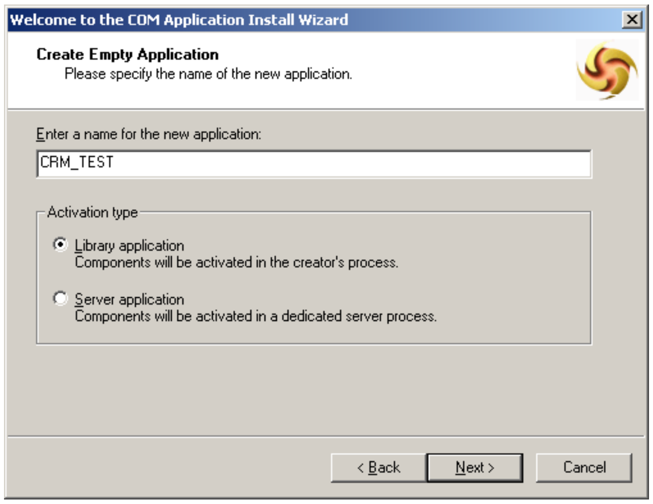
`Settings->Control Panel->Administrative Tools->Component Services`

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

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

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

Figure 30 CRM_TEST Application



Welcome to the COM Application Install Wizard

Create Empty Application
Please specify the name of the new application.

Enter a name for the new application:
CRM_TEST

Activation type
[● Library application]
Components will be activated in the creator's process.
○ Server application
Components will be activated in a dedicated server process.

6 Click Next to continue, and then Finish.

7 Expand the CRM_TEST component, right click on the components folder, click New Component. The COM Component Install Wizard opens. Click Next to continue.

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

4.3 Sample Schema Implementation

The sample implementation is available in the **samples** directory of the e*Gate CD-ROM. Navigate to the directory

samples/jmsapi:

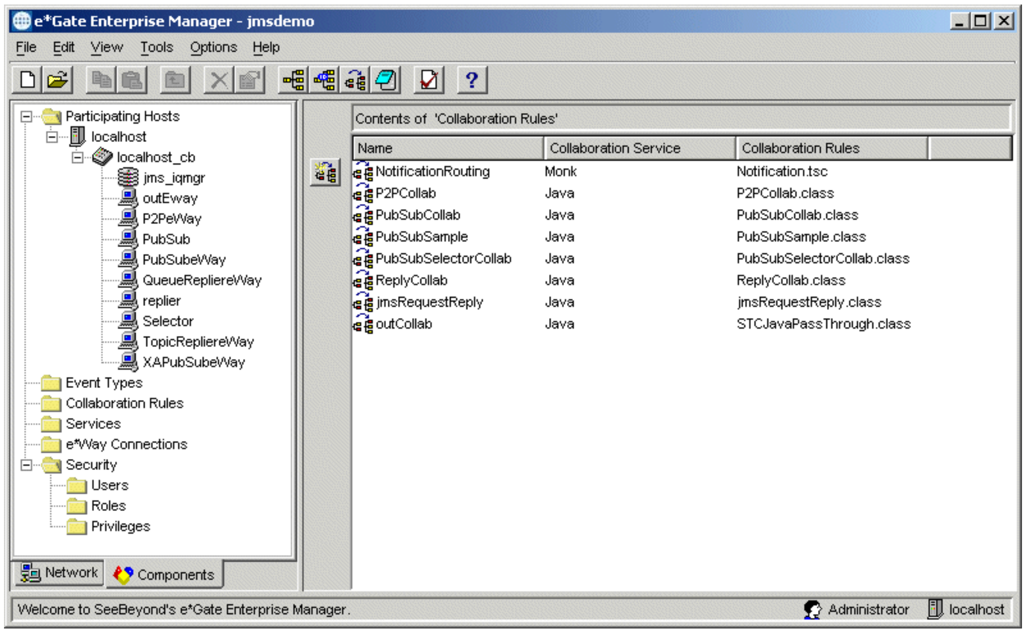
and for further instructions, see the README.html in the directory to install.

You can import the schema at the startup of the e*Gate Enterprise Manager, or by selecting “New Schema” from the File menu, once the e*Gate Enterprise manager has opened. For either case, select “Create from export:” and navigate to the .zip file containing the necessary sample.

4.3.1 e*Gate Sample JMS Schema Overview

The schema is designed to be combined with the code samples provided to separate paradigm behavior into individual e*Ways.

Figure 31 JMS Sample Schema Enterprise Manager View



Contents of 'Collaboration Rules'

Name	Collaboration Service	Collaboration Rules
NotificationRouting	Monk	Notification.tsc
P2PCollab	Java	P2PCollab.class
PubSubCollab	Java	PubSubCollab.class
PubSubSample	Java	PubSubSample.class
PubSubSelectorCollab	Java	PubSubSelectorCollab.class
ReplyCollab	Java	ReplyCollab.class
jmsRequestReply	Java	jmsRequestReply.class
outCollab	Java	STCJavaPassThrough.class

Welcome to SeeBeyond's e*Gate Enterprise Manager.

The e*Gate schema created to perform with the above code sample contains the following:

- SeeBeyond JMS IQ Manager
- 13 Event Types
- 1 Event Type Definition
- 2 JMS e*Way Connections
- 8 Java Collaboration Rules
- 2 Multi-mode e*Ways
- 7 File e*Ways
- 8 Java Collaborations

SeeBeyond JMS IQ Manager

The IQ Manager defaults to the SeeBeyond JMS IQ Manager. For more information see the *SeeBeyond JMS Intelligent Queue User's Guide*.

Event Type

When creating the Event Type, the name of the Event must correspond to the Topic or Queue being used in the code sample. For the samples provided, a Topics or Queues are created for each of the demonstration. The number of Event Types created is dependant on the number of desired Topics or Queues to be used.

Event Type Definition

For the sample, no specific ETDs have been created, a single node .xsc (root.xsc) has been used. Whether or not you require a specific ETD will depend completely on the parsing intended. For more information on creating ETDs, see the *e*Gate Integrator User's Guide*.

JMS e*Way Connections

In the sample schema provided, e*Way Connections are created to communicate with the external system, and configured. It is important to set the Connection Type (Topic or Queue), the expected Output Message Type (bytes or text), and ensure that both values correspond in the code. The additional parameters values are left to default (in the sample). For more information on configuring JMS e*Way Connections, see the **"JMS e*Way Connection Parameters" on page 45**.

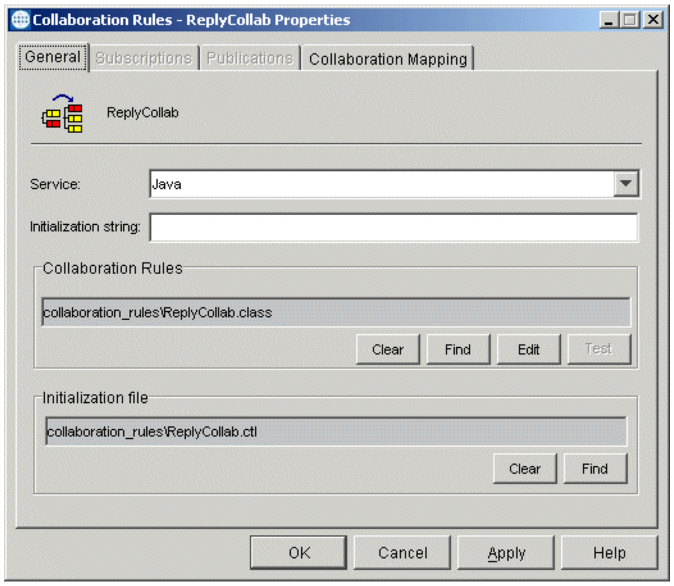
Java Collaboration Rules

In the sample schema, a Collaboration, and a separate Java Collaboration Rule are created for each sample e*Way.

Note: *The read and write method calls must correspond to the expected data types. For example, if the Message Type is set for byte, the corresponding methods would be one of the readByte and writeByte methods.*

For **jmsRequestReply**, the Java Service is selected.

Figure 32 jmsRequestReply


Collaboration Rules - ReplyCollab Properties

General | Subscriptions | Publications | Collaboration Mapping

ReplyCollab

Service: Java

Initialization string:

Collaboration Rules

collaboration_rules\ReplyCollab.class

Initialization file

collaboration_rules\ReplyCollab.ctl

For the Collaboration Mapping, the Instance Names “In” and “Out” are designated respectively. The **root.xsc** ETD is assigned to both. The trigger is set for the inbound instance, while Manual Publish is set for the outbound. Return to the General Tab, and select **New** for the Collaboration Rule. The Collaboration Editor opens showing the Source Event as the created “In” instance, with the “Out” instance as the Destination Event.

Create a rule below the retBoolean variable. To enable the “Reply” functionality, add the following line in the Rule dialog box:

```
String topic = getin().readProperty("JMSReplyTo");
```

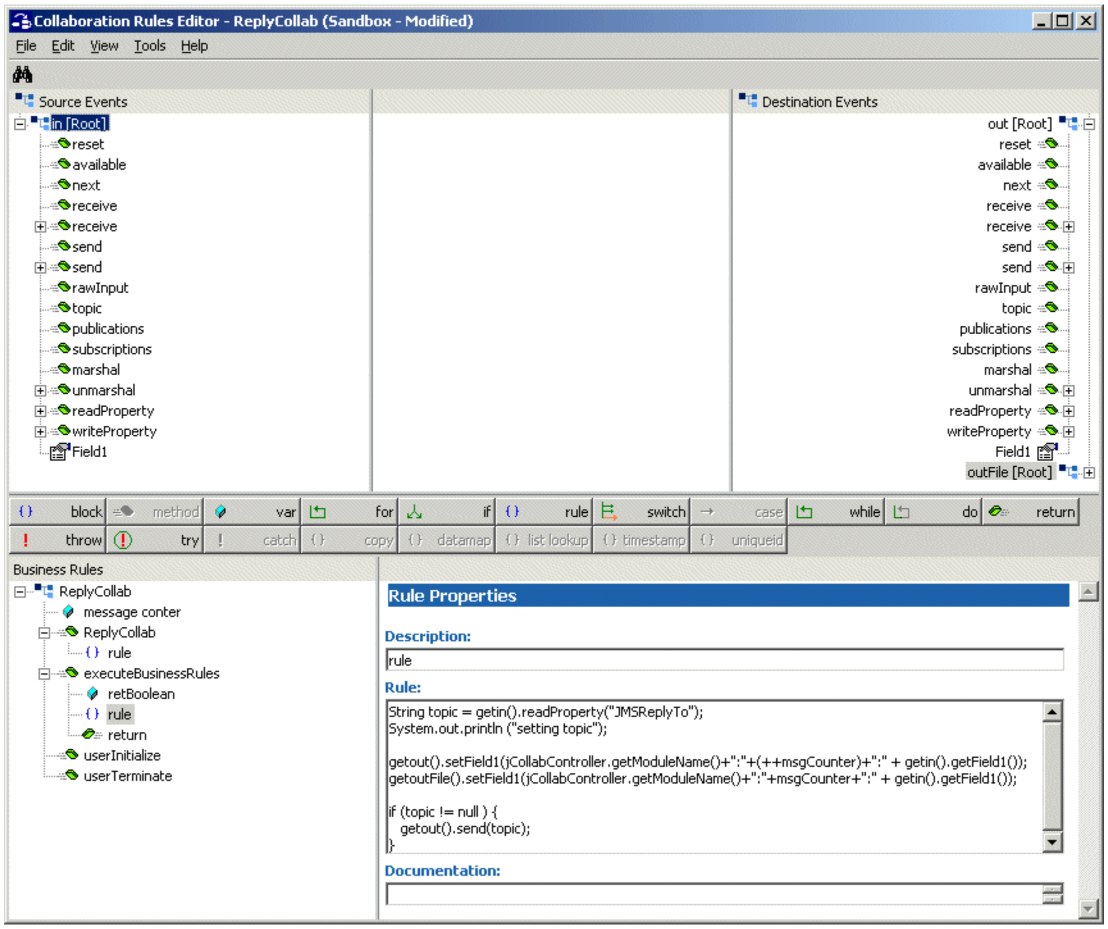
This line obtains the readProperty (“JMSReplyTo”) from the inbound TopicRequestor, providing a “return address”.

The additional functionality was added:

```
getout().setField1(jCollabController.getModuleName()+":"+(++msgCounte
r)+":" + getin().getField1());

getoutFile().setField1(jCollabController.getModuleName()+":"+msgCount
er+":" + getin().getField1());

if( topic != null )
  getout().send(topic);
```

The sample provides one scenario. The Collaboration Rule appears below:

Figure 33 ReplyCollab Rule



Description: rule

Rule:

```
String topic = getin().readProperty("JMSReplyTo");
System.out.println ("setting topic");

getout().setField1(jCollabController.getModuleName()+":"+(++msgCounter)+":" + getin().getField1());
getoutFile().setField1(jCollabController.getModuleName()+":"+msgCounter+":" + getin().getField1());

if (topic != null ) {
   getout().send(topic);
}
```

Compile, Save and Promote to runtime.

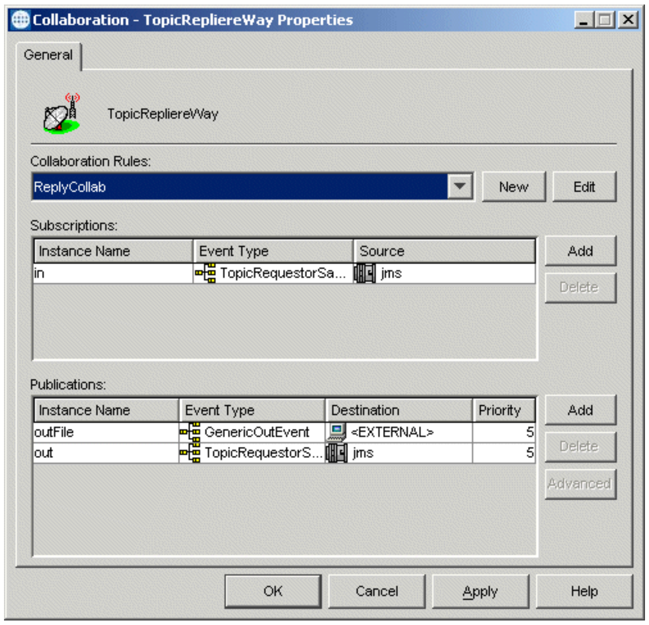
Multi-mode e*Way

The Multi-mode e*Way (replier and outEway), along with the e*Way Connection, provides connectivity to the external system via the SeeBeyond JMS IQ Manager. The executable required is stceway.exe. For more information about configuring the Multi-mode e*Way, see the *Standard e*Way Intelligent Adapter User's Guide*.

Java Collaboration

Associated with the e*Way, the Collaboration designates the functionality as defined in the Collaboration Rule with the e*Way. The Collaboration displays the Event Type and Source for both the Subscription(s) and Publication(s). It is very important that these values be set according to the expected behavior. In the sample, the Collaboration, TopicRepliereWay appears below:

Figure 34 TopicReplierеWay Collaboration


Collaboration - TopicReplierеWay Properties

General

TopicReplierеWay

Collaboration Rules: ReplyCollab

Subscriptions:

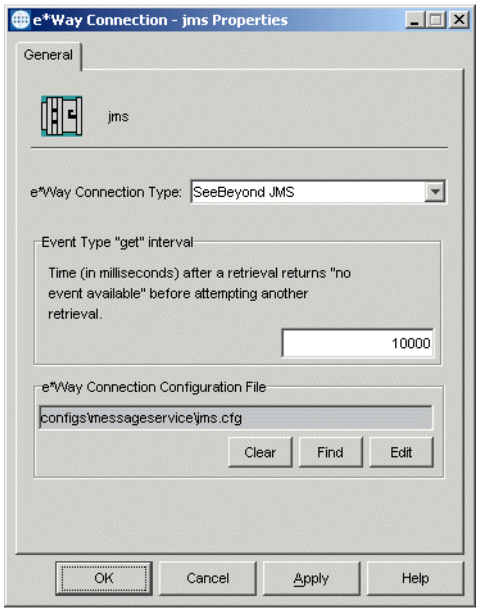
Instance Name	Event Type	Source
in	TopicRequestorSa...	jms

Publications:

Instance Name	Event Type	Destination	Priority
outFile	GenericOutEvent	<EXTERNAL>	5
out	TopicRequestorS...	jms	5

By selecting the jmsRequestReply Collaboration Rule, the Instance Names, Event Types the Source and the Destination for both the Subscription and Publication (as related to the Collaboration), the schema is now ready to execute.

Figure 35 e*Way Connection


e*Way Connection - jms Properties

General

jms

e*Way Connection Type: SeeBeyond JMS

Event Type "get" interval

Time (in milliseconds) after a retrieval returns "no event available" before attempting another retrieval.

10000

e*Way Connection Configuration File

configs\messageservice\jms.cfg

4.3.2 Executing the Schema

From the command line start the Control Broker.

```
stccb.exe -rh <host> -rs <schema_name> -un Adminstrator -up STC -ln
localhost_cb
```

At this point the schema will auto-start all the components. The external code provided must be compiled and run, making sure that the host name and port number points to the Participating Host, on which the JMS IQ Manager is running.

Chapter 5

Configuring the Multiplexer e*Way

5.1 Configuring the Multiplexer Client

Once the e*Gate API Kit has been installed successfully, additional steps are required to finish the setup, before data exchange can begin. Both the multiplexer client, which represents the machine upon which the external application resides, and the multiplexer server, which is the machine upon which the Participating Host resides, require handling. In this section, the steps are included for setting up the Multiplexer.

5.1.1 Considerations

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

1. Install the required client files on the external system.
2. Configure the client components as necessary to use the TCP/IP port specified above in “Push IP Port.”

5.1.2 Setting up the Multiplexer

To begin using the Multiplexer, do the following:

Copy the **stc_xipmpclnt.dll**, **stc_common.dll**, **stc_ewipmpclnt.dll** files from your eGate\client\bin to a directory on your external system.

From the command prompt of the external system, register all three files **stc_xipmpclnt.dll**, **stc_common.dll**, **stc_ewipmpclnt.dll** into the Windows NT or Windows 2000 registry by doing the following:

regsvr32 your_path_location\<file_name>

5.1.3 Setting up the Muxpooler

To begin using the Muxpooler, do the following:

Copy the stc_common.dll and stcph.jar files from your eGate\client\bin to a directory on your external system.

From the command prompt of the external system, register stc_common.dll in the Windows NT or 2000 registry with the following command:

regsvr32 your_path_loaction\<file_name>

And include stcph.jar in your classpath.

This chapter describes the e*Way configuration parameters and the external configuration requirements for the e*Gate API Kit.

Important: *From the perspective of the e*Gate GUIs, the e*Gate API Kit e*Way is not a system of components distributed between the web server and a Participating Host, but a single component that runs an executable file (the multiplexer* ***stcewipmp.exe****). When this manual discusses procedures within the context of any e*Gate GUI (such this chapter, which deals in part with the e*Way Editor), the term "e*Way" refers only to the Participating Host component of the e*Way system.*

5.2 Configuring the Multiplexer Server

5.2.1 Multiplexer e*Way Configuration Parameters

e*Way configuration parameters are set using the e*Way Editor.

To change e*Way configuration parameters:

1. In the Enterprise Manager's Component editor, select the e*Way you want to configure and display its properties.
2. Under **Configuration File**, click **New** to create a new file, **Find** to select an existing configuration file, or **Edit** to edit the currently selected file.
3. In the **Additional Command Line Arguments** box, type any additional command line arguments that the e*Way may require, taking care to insert them *at the end* of the existing command-line string. Be careful not to change any of the default arguments unless you have a specific need to do so.

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

The e*Way's configuration parameters are organized into a single section: General Settings.

General Settings

The parameters in this section specify the name of the external client system and the IP port through which e*Gate and the client system communicates.

Request Reply IP Port

Description

Specifies the IP port that the e*Way will listen (bind) for client connections. This parameter is used for Request/Reply behavior.

Required Values

A valid TCP/IP port number between 1 and 65536. The default is **26051**. Normally, you only need to change the default number if the specified TCP/IP port is in use, or you have other requirements for a specific port number.

Push IP Port

Description

Specifies the IP port through which this e*Way allows an external system to connect and receive unsolicited (without submitting a request) Events.

Required Values

A valid TCP/IP port number between 0 and 65536. The default is **0**.

Additional Information

Any Event that this e*Way receives that has zero values for all fields in the 24 byte MUX header is sent to all callers of the **WaitForUnsolicited**. This parameter is optional. If set to zero, the e*Way will follow the Request/Reply scenario and not accept unsolicited Events.

Rollback if no Clients on Push Port

Description

Specifies whether the Event will continually roll back if there are no push clients connected.

Required Values

Yes or **No**. If set to **Yes**, the Event will continually roll back if there are no push clients connected.

Wait For IQ Ack

Description

Specifies whether the send client function does NOT return until the Event is committed to the IQ.

Required Values

Yes or **No**. If set to **Yes**, the send client function does NOT return until the Event is committed to the IQ.

Caution: *This parameter should be set if the data must be committed to the IQ on every transaction before the API returns to the client. Setting this parameter to Yes will significantly impact performance. If normal request/reply type transactions are being sent/received, and the data can be recreated at the client, this parameter should not be set.*

Send Empty MSG When External Disconnect

Description

Specifies whether the e*Way sends an empty incoming message (containing only the multiplexer header) when an external client disconnects.

Required Values

Yes or **No**. If set to **Yes**, the e*Way sends an empty incoming message when an external client disconnects.

MUX Instance ID

Description

Specifies whether the specified 8 (eight) bytes is prepended to the 24 (twenty-four) byte session ID of the request received from the external connection before sending to e*Gate.

Required Values

A string. If this value is other than "0", the 8 bytes are prepended to the 24 byte session ID. The default is **0**.

Note: *This is a string where "00" and "00000000" are valid MUX Instance IDs, while "0" is to turn this option off. Only the first 8 bytes are used.*

MUX Recovery ID

Description

Specifies whether the 8 bytes are prepended to the reply and republish back to e*Gate provided the value is other than "0" and the multiplexer finds that the session related to the MUX ID in the return message has been dropped.

Required Values

```
A string. If this value is other than "0", the 8 bytes are
prepended to the 24 byte session ID. The default is 0.
```

Note: *This is a string where "00" and "00000000" are valid MUX Recovery IDs, while "0" is to turn this option off. Only the first 8 bytes are used.*

Chapter 6

Implementing the Multiplexer e*Way

This chapter describes the e*Way configuration parameters, the external configuration requirements, and the implementation for the e*Gate API Kit Multiplexer e*Way.

Important: *From the perspective of the e*Gate GUIs, the e*Gate API Kit e*Way is not a system of components distributed between the web server and a Participating Host, but a single component that runs an executable file (the multiplexer **stcewipmp.exe**). When this manual discusses procedures within the context of any e*Gate GUI (such this chapter, which deals in part with the e*Way Editor), the term "e*Way" refers only to the Participating Host component of the e*Way system.*

6.1 Implementing the Multiplexer Models

The e*Gate API kit Multiplexer supports three basic architectures:

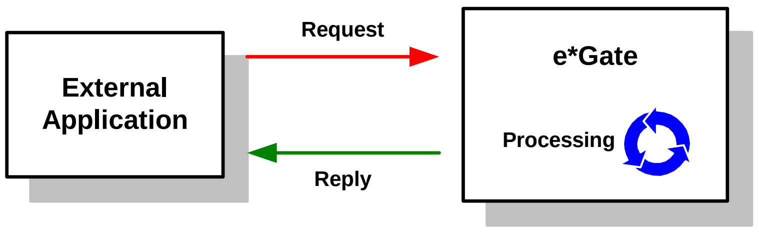
1. Request/Reply, where data is sent to the e*Gate system and a response is returned
2. Send-only, where data is sent to the e*Gate system but no data is returned
3. Receive, where an external system connects to the e*Gate system and allows for the delivery unsolicited Events

This section discusses how to use the Multiplexer to exchange data with an e*Gate system.

6.1.1 Multiplexer Overview

Request Reply

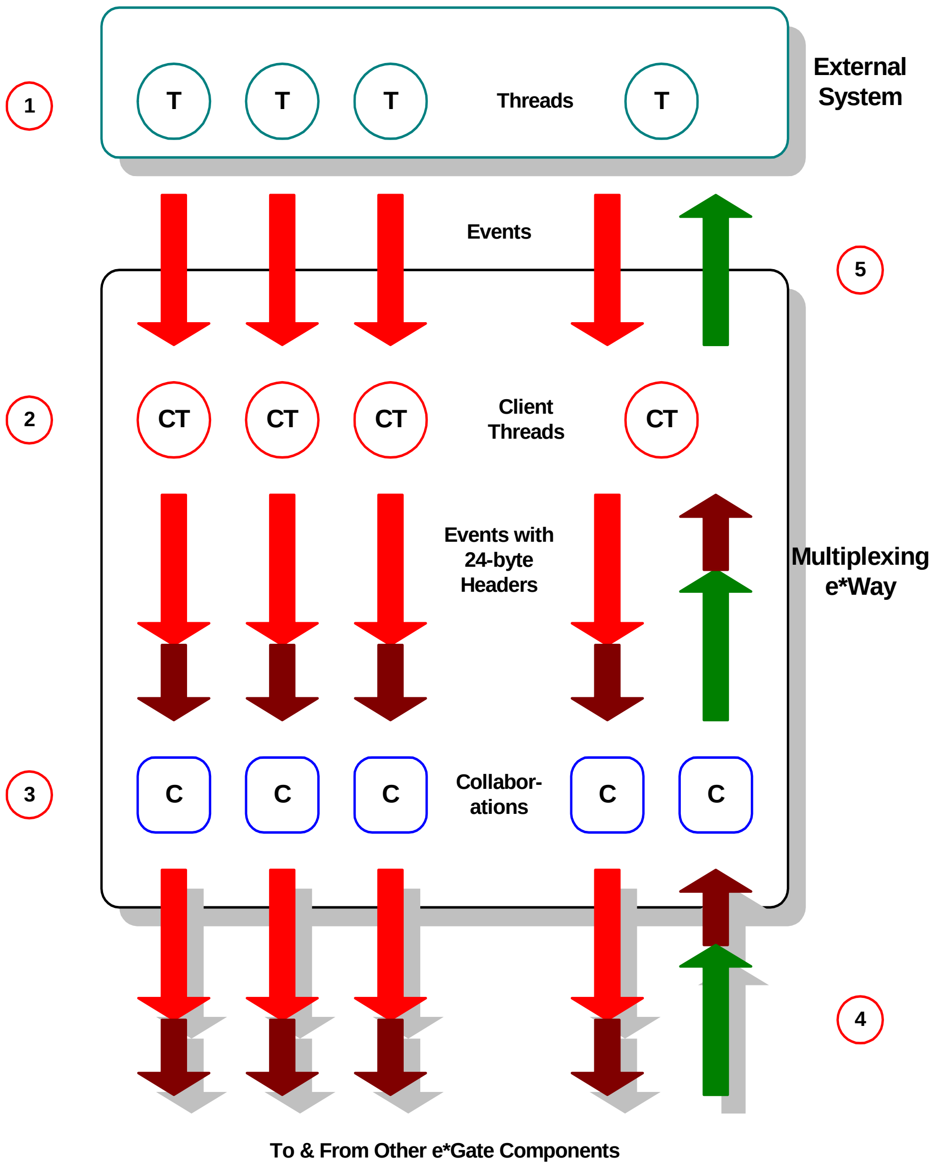
Figure 36 The Multiplexer concept



The external system uses API-kit client components to send the data to the multiplexing e*Way using an SeeBeyond-proprietary IP-based protocol. Depending on the external system's requirements and capabilities, these client components can be single- or multi-threaded.

Figure 37 illustrates how the multiplexing e*Way receives data from an external application and returns processed data to the same application.

Figure 37 Data flow through the Multiplexing e*Way



Client threads within the e*Way package the data as e*Gate Events, adding a 24-byte header. Among other functions, this header provides "return address" information that can optionally be used to return data to the client thread that originated it.

Each e*Way can handle up to 1,000 client threads at once. If your requirements demand more processing power, you can define more multiplexing e*Ways.

Collaborations within the e*Way perform any appropriate processing that may be required, and route the processed Events to other destinations.

Note: *The 24-byte header **must** be preserved as the Events are processed through the e*Gate system.*

The e*Way can also route information back to the thread on the external system that sent the original data.

Processed data, still containing the original 24-byte header, is returned to the multiplexing e*Way.

The e*Way uses the 24-byte "return address" to identify the destination of the data to be returned to the external system.

The e*Way returns the data, minus the 24-byte header, to the external system.

6.1.2 Multiplexer Request/Reply Sample Schema

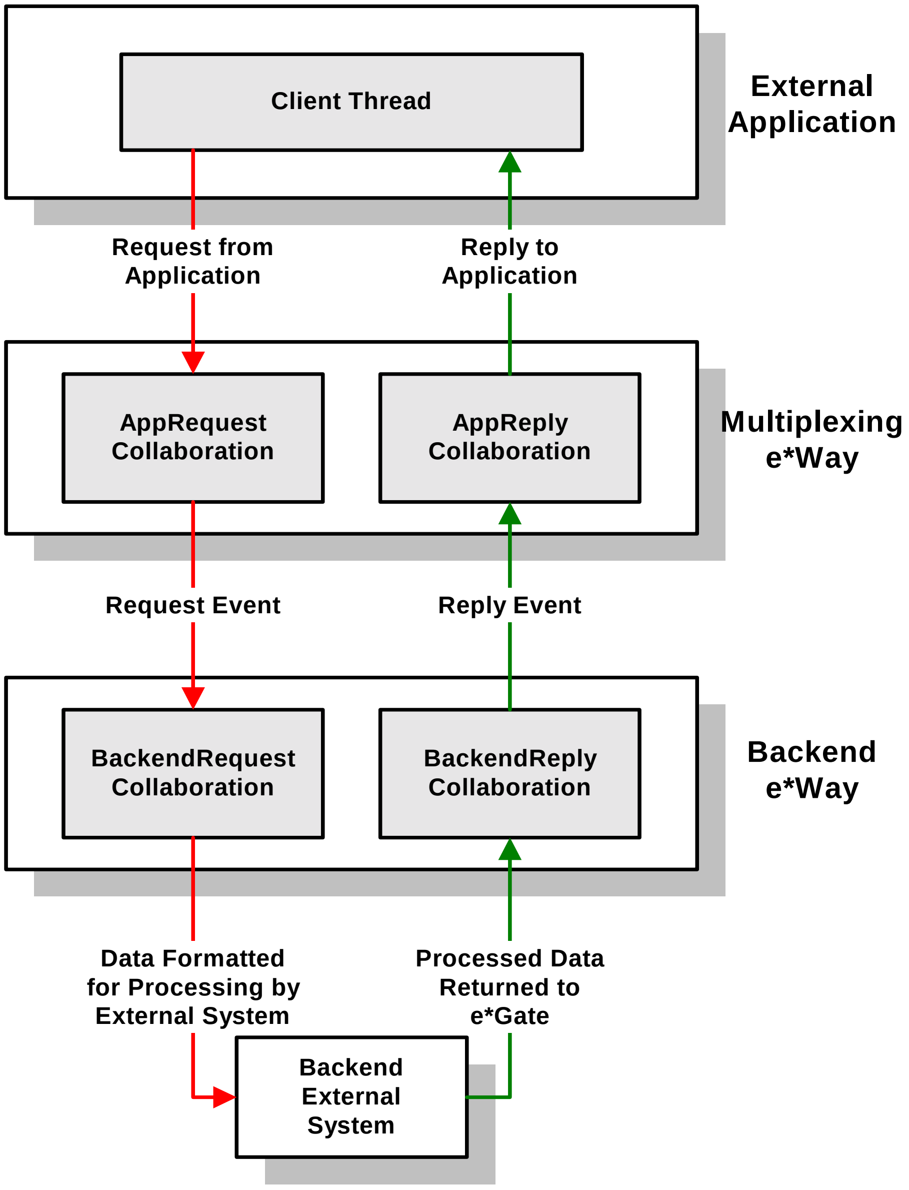
Request/Reply schemas have two classes of components:

1. “Front end” components that handle communications with the external application. These components receive requests and route replies to the correct destination.
2. “Back end” components that process the requests and compose the replies. These components also provide the bridge between the e*Gate system and your existing systems.

The multiplexing e*Way and its related Collaborations comprise the front-end components. Additional e*Ways and their related Collaborations comprise the back-end components. The backend e*Way(s) can be of any type required to communicate with the external system(s).

Figure 38 below illustrates a Request/Reply schema.

Figure 38 The Request/Reply schema



1 Data enters the e*Gate system through the multiplexing e*Way via the **Request** Collaboration.

2 The **Request** Collaboration publishes the Request Event.

3 The **BackendRequest** Collaboration within the back-end e*Way subscribes to the Request Event, and routes or processes the data as appropriate.

4 After the data has been processed, the back-end e*Way's **BackendReply** Collaboration publishes the data as the Reply Event.

5 The **Reply** Collaboration within the multiplexing e*Way subscribes to the Reply Event.

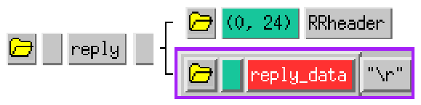
6 The multiplexing e*Way returns the processed data to the requesting thread in the external application.

6.1.3 ETDs, Collaboration Rules, and the "Return Address" Header

As discussed in **"Request-Reply" on page 14**, the multiplexing e*Way maintains "return address" information in a 24-byte header that must be preserved as the data flows through the e*Gate system.

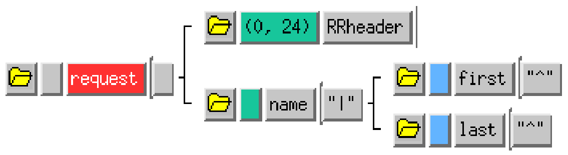
The simplest Event Type Definition (ETD) that can be used within a Request/Reply schema has two nodes: one for the header, the second for the remainder of the Event data.

Figure 39 The simplest Request/Reply ETD



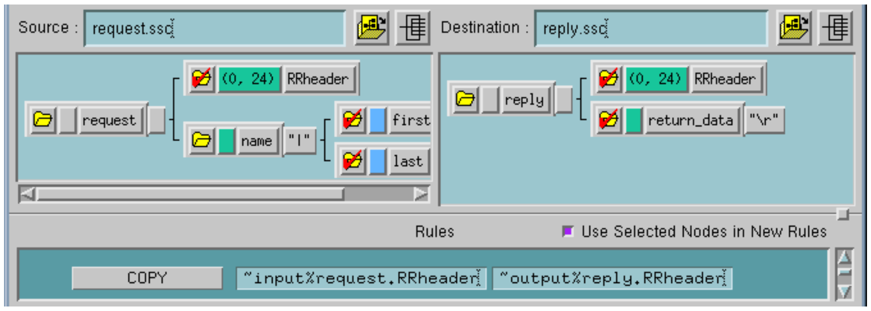
This ETD is sufficient if you wish to send data through the e*Gate system simply as a blob. If your data has a more complex structure, add subnodes to the "data" node, then describe the structure of the data within those subnodes. Figure 39 below illustrates an ETD that describes delimited data (for example, as in the data "First name^Last name").

Figure 40 A Request/Reply ETD for delimited data



Collaboration Rules that manipulate data between ETDs must preserve the Request/Reply header (in the figures above, "RRheader"). Be sure that each Collaboration Rule that manipulates Request/Reply data copies the contents of the Request/Reply header from the source ETD to the target ETD (as shown in Figure 41 below).

Figure 41 Copying the Request/Reply header



```
COPY    ~input%request.RRheader    ~output%reply.RRheader
```

6.1.4 Using the C APIs

The C application must do the following:

1. Load the following header files (located on the installation CD in the SDK folder at root)

    ```
    gendefs.h, tracelog.h, ewipmpclnt.h
    ```

2. Link the stc_ewipmpclnt.dll and the stc_common.dll at compile time.
3. Use the EWIPMP_Open function to open a connection to the multiplexer e*way.
4. Get the data from the user.
5. Format the data as appropriate to be processed by e*Gate.
6. Use the EWIPMP_Send function to send data to the e*Gate system.
7. Use the EWIPMP_Wait function to cause execution to pause long enough for e*Gate to process and return the data.
8. Use the EWIPMP_Free function to free the memory associated with the returned data in the message buffer.
9. Close the connection using the EWIPMP_Close function

6.1.5 Using the Java APIs

The Java application must do the following:

1. Load the **com.stc.ewip** package
2. Create an instance of the IPMPReqReply “mux” object
3. Define the host name, TCP/IP port, expiration time, and timeout using the “set” methods (described beginning with **“setHost” on page 40**)
4. Use the connect method (see **“connect” on page 31**) to open a connection to the multiplexing e*Way
5. Get the data from the user
6. Assemble the data to be sent to e*Gate in an appropriate format

7 Use the sendMessage method (see **"sendMessage" on page 39**) to send the request to the e*Gate system

8 Use one of the getResponse methods (such as **"getResponse" on page 35**) to retrieve the response from the e*Gate system.

9 Close the connection using the disconnect method (see **"disconnect" on page 32**)

A commented sample.java file is available on the e*Gate installation CD-ROM; see **"Sample Implementation" on page 114** for more information.

6.1.6 Using the ActiveX Control Within VBasic Applications

The Visual Basic application must do the following:

1 Create an instance of the ActiveX "MUX" object

2 Define the host name and TCP/IP port numbers

3 Use the Connect method (see **"Connect" on page 20**) to open a connection to the e*Gate system

4 Get the data from the user

5 Format the data as appropriate to be processed by e*Gate

6 Use the Send method (see **"Send" on page 27**) to send data to the e*Gate system

7 Use the Wait method (see **"Wait" on page 28**) to cause the executing thread to pause long enough for e*Gate to process and return the data

8 Use one of the "ReplyMessageAs" methods (such as **"ReplyMessageAsString" on page 25**) to display the returned data

9 Handle errors using one of the "LastError" methods (such as **"LastErrorCode" on page 22**)

10 Close the connection using the Disconnect method (see **"Disconnect" on page 21**)

Additional information can be found in commented sample files (see **"Sample Implementation" on page 114** for more information).

6.1.7 Using Perl APIs

The Perl script must do the following:

1 Use the Multiplexer_Init subroutine (see **"Multiplexer_Init" on page 67**) to specify the location of the **stc_ewipmpclntperl.pm** and **stc_ewipmpclntjperl.so** files

2 Define the host name and TCP/IP port numbers

3 Format the user data as appropriate for processing within e*Gate

4 Use the Multiplexer_Open subroutine (see **"Multiplexer_Open" on page 68**) to open a connection to the e*Gate Participating Host

5 Use the Multiplexer_Send subroutine (see **"Multiplexer_Send" on page 69**) to send data to the e*Gate Participating Host

6. Use the Multiplexer_Wait subroutine (see **"Multiplexer_Wait" on page 71**) to cause the Perl script to pause long enough for e*Gate to process and return the data
7. Use the Multiplexer_ToString subroutine (see **"Multiplexer_ToString" on page 70**) to obtain the returned data and display it within the user's browser
8. Use the Multiplexer_Free subroutine (see **"Multiplexer_Free" on page 66**) to free the memory associated with the returned data
9. Close the connection using the Multiplexer_Close subroutine (see **"Multiplexer_Close" on page 65**)

6.2 Using the Cobol APIs

The following code demonstrates a sample set of actions:

1. The MUXxxx load module must be included in the link step when the calling program is compiled. For CICS only: the CICS IP socket routines should be included in the same link step.
2. Call "MUXxxx" with the appropriate parameters to establish a connection to the multiplexer e*Way.
3. Call "MUXxxxS" to SEND data to e*Gate, passing the data and its length as specified in the parameter list.
4. Call "MUXxxxR" to RECEIVE data from e*Gate; the length of the data received is returned by the API. Use the MUXAPI-hsecs-to-wait parameter to cause the execution to pause long enough for e*Gate to process and return the data.
5. Repeat the SEND and RECEIVE as desired to continue passing and receiving data.
6. Call "MUXxxxC" to close the connection.

Note: *Where MUXxxx is MUXAPI for CICS, MUXIMS for IMS, and MUXBAT for Batch.*

Note: *Once the connection has been opened successfully, if any of the subsequent functions fail, the connection must be closed before continuing.*

The following Cobol "client" program illustrates a simple Open-Send-Receive-Close scenario, in which a seventeen character text message (hard-coded in working storage in this example), is sent to the e*Gate "server", and waits one second to receive a response.

Note: *Where in the following program, MUXxxx is MUXAPI for CICS, MUXIMS for IMS, and MUXBAT for Batch.*

```
000011 Identification Division.
000012*===============================================================*
000013 Program-id.  MUXCLI.
000014
000015*===============================================================*
000016 Environment Division.
```

```
000017*================================================================*
000018
000019*================================================================*
000020 Data Division.
000021*================================================================*
000022
000030 WORKING-STORAGE SECTION.
000031
000670*================================================================*
000680* Variables used for the MUXAPI function calls                  *
000690*================================================================*
000691
000693 01  MUXAPI-handle           pic s9(8)  binary value +0.
000694* move host name to this field:
000695 01  MUXAPI-remote-host      pic  x(24) value 'remote.host.name'.
000696* default port:
000697 01  MUXAPI-remote-port      pic  9(8)  binary value 26051.
000698 01  MUXAPI-message-len      pic  9(8)  binary.
000700 01  MUXAPI-secs-to-expire   pic  9(8)  binary.
000701 01  MUXAPI-returnmsg-len    pic  9(8)  binary.
000703 01  MUXAPI-hsecs-for-ack pic  9(8)  binary value 100.
000710 01  MUXAPI-errno            pic  9(8)  binary value 0.
000711 01  MUXAPI-retcode          pic s9(8)  binary value +0.
000712
000713*================================================================*
000714* misc
000720*================================================================*
000730
000740 01  test-message            pic x(17) value 'Hello From MUXCLI'.
000741 01  MUXAPI-message          pic  x(32727) value spaces.
000742 01  MUXAPI-returnmsg        pic  x(32727) value spaces.
000750
000760*================================================================*
002002 PROCEDURE DIVISION.
002003*================================================================*
002004
002005 Main.
002010     perform MUXAPI-open-connection
002011     if MUXAPI-retcode < 0
002012         go to exit-program
002013     end-if
002014
002015     move test-message to MUXAPI-message
002016     move 17 to MUXAPI-message-len
002017
002020     perform MUXAPI-send-message
002022     if MUXAPI-retcode >= 0
002030         perform MUXAPI-receive-response
002040     end-if
002041
002050     perform MUXAPI-close-connection.
002051
002052 exit-program.
002053     exec CICS return
002060     end-exec
002061     exit program.
002070
002090 MUXAPI-open-connection.
002102     call "MUXxxx" using
002105          MUXAPI-handle
002106          MUXAPI-remote-host
002107          MUXAPI-remote-port
002108          MUXAPI-errno
002109          MUXAPI-retcode.
002200
005200 MUXAPI-send-message.
005221     call "MUXxxxS" using
005223          MUXAPI-handle
005240          MUXAPI-message-len
005241          MUXAPI-message
005243          MUXAPI-hsecs-for-ack
005244          MUXAPI-errno
```

```
005250          MUXAPI-retcode.
005291
005300 MUXAPI-receive-response.
005311     call "MUXxxxR" using
005313          MUXAPI-handle
005314          MUXAPI-returnmsg-len
005315          MUXAPI-returnmsg
005318          MUXAPI-hsecs-to-wait
005319          MUXAPI-errno
005320          MUXAPI-retcode.
005330
005500 MUXAPI-close-connection.
005503     call "MUXxxxC" using
005505          MUXAPI-handle
005506          MUXAPI-errno
005509          MUXAPI-retcode
```

6.3 Sample Implementation

A sample implementation is available in the **samples** directory of the e*Gate CD-ROM. Navigate to the directory

samples/ewmux

and follow the directions in the README file in that directory.

In the demonstration schema, the back end is provided by a TCP/IP e*Way that applies data-manipulation Collaboration Rules and a Loopback e*Way that sends the TCP/IP e*Way's output back into the e*Gate system.

If you use the Enterprise Manager to examine the sample schema, note that the Loopback e*Way has no Collaborations; the Loopback e*Way requires none to perform its "loopback" function.

Note: *The TCP/IP e*Way used in the demonstration schema was developed specifically for this use. A general-purpose TCP/IP e*Way is also available for other uses; contact SeeBeyond for more information.*

Chapter 7

Client Libraries for the e*Gate Message Service

7.1 The Java APIs

The JMS API Javadocs can be downloaded from:

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

7.2 Supported Java Message Service (JMS) Classes

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

The current implementation of JMS Java APIs within the e*Gate API Kit support the following classes:

7.2.1 com.seebeyond.jms.client.STCTopicRequestor

Helper class to simplify making service requests.

Methods of the STCTopicRequestor Object

The STCTopicRequestor Method

Constructs the TopicRequestor.

```
STCTopicRequestor(Session session, Topic topic)
```

Name	Description
session	The name of the topic session.
topic	The name of the topic.

The Request Method

Send a request and wait for a reply

```
public com.seebeyond.jms.client.Message
    request(com.seebeyond.jms.client.Message message)
    throws com.seebeyond.jms.client.JMSException
```

Name	Description
message	The message text.

Throws

com.seebeyond.jms.client.JMSException

The Request Method

Send a request and wait for a reply

```
public com.seebeyond.jms.client.Message
    request(com.seebeyond.jms.client.Message message long l)
    throws com.seebeyond.jms.client.JMSException
```

Name	Description
message	The message text.
l	The amount of time to wait for a message in milliseconds.

The CloseMethod

Since a provide may allocate resources on behalf of an STCTopicRequestor outside of the JMV, clients should close them, when they are not needed.

```
void close( )
    throws com.seebeyond.jms.client.JMSException
```

7.2.2 com.seebeyond.jms.STCQueueRequestor

Helper class to simplify making service requests.

Methods of the STCQueueRequestor Object

The STCQueueRequestor Method

Construct the QueueRequestor.

```
STCQueueRequestor(com.seebeyond.jms.client.QueueSession queuesession,
com.seebeyond.jms.client.Queue queue1)
```

Name	Description
queuesession	The QueueSession.
queue1	Queue name.

The Request Method

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

```
public com.seebeyond.jms.client.Message
    request(com.seebeyond.jms.client.Message message)
   throws com.seebeyond.jms.client.JMSException
```

Name	Description
message	The message.

The Request Method

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

```
public com.seebeyond.jms.client.Message
    request(com.seebeyond.jms.client Message message, long l)
```

Name	Description
message	The message.
l	The amount of time to wait for a message in milliseconds.

7.2.3 class javax.jms.JMSException

```
public class JMSException
   extends java.lang.Exception
```

This is the root class of all JMS API exceptions.

This class provides the following information:

- A provider-specific string describing the error. This string is the standard exception message and is available via the getMessage method.
- A provider-specific string error code
- A reference to another exception. Often a JMS API exception will be the result of a lower-level problem. If appropriate, this lower-level exception can be linked to the JMS API exception.

The JMSException Method

Construct a JMSException with reason and errorCode for the exception.

```
public JMSException(java.lang.String reason,
                    java.lang.String errorCode)
```

Name	Description
reason	A description of the exception
errorCode	A string specifying the vendor specific error code.

The JMSException Method

Construct a JMSException with a reason and with error code defaulting to null.

```
public JMSException(java.lang.String reason)
```

Name	Description
reason	A description of the exception

The getErrorCode Method

Gets the vendor specific error code.

```
public java.lang.String getErrorCode()
```

The getLinkedException

Gets the exception linked to this exception.

```
public java.lang.Exception getLinkedException()
```

The setLinkedException

Adds a linked exception.

```
public void setLinkedException(java.lang.Exception ex)
```

Name	Description
ex	The linked exception.

7.2.4 class javax.jms.IllegalStateException

```
public class IllegalStateException
   extends JMSException.
```

This exception is thrown when a method is invoked at an illegal or inappropriate time or if the JMS IQ server is not in an appropriate state for the requested operation.

The IllegalStateException Method

Constructs an IllegalStateException with reason and errorCode for excpetion.

```
public IllegalStateException(java.lang.String reason,
                             java.lang.String errorCode)
```

Name	Description
reason	A description of the exception
errorCode	A string specifying the vendor specific error code.

The IllegalStateException Method

Constructs an IllegalStateException with reason. Error code defaults to null.

```
public IllegalStateException(java.lang.String reason)
```

7.2.5 class.javax.jms.InvalidClientIDException

```
public class InvalidClientIDException
   extends JMSException.
```

This exception is thrown when a client attempts to set a connection's client ID to a value that is rejected by JMS IQ server.

The InvalidClientIDException

Constructs an InvalidClientIDException with reason and errorCode for excpetion.

```
public InvalidClientIDExceptin(java.lang.String reason,
                               java.lang.String errorCode)
```

Name	Description
reason	A description of the exception
errorCode	A string specifying the vendor specific error code.

The InvalidClientIDException

Constructs an InvalidClientIDException with reason. The error code defaults to null.

```
public InvalidClientIDException(java.lang.String reason)
```

Name	Description
reason	A description of the exception

7.2.6 class javax.jms.InvalidDestinationException

```
public class InvalidDestinationException
   extends JMSException.
```

This exception is thrown when a destination either is not understood by the JMS IQ server or is no longer valid.

The InvalidDestinationException Method

Constructs an InvalidDestinationException with reason and errorCode for excpetion.

```
public InvalidDestinationException(java.lang.String reason,
                                   java.lang.String errorCode)
```

Name	Description
reason	A description of the exception
errorCode	A string specifying the vendor specific error code.

The InvalidDestinationException Method

Constructs an InvalidDestinationException with reason. The error code defaults to null.

```
public InvalidDestinationException(java.lang.String reason)
```

Name	Description
reason	A description of the exception

7.2.7 class javax.jms.InvalidSelectorException

```
public class InvalidSelectorException
   extends JMSException.
```

This exception is thrown when a JMS client attempts to give the JMS IQ server a message selector with invalid syntax.

The InvalidSelectorException Method

Constructs an InvalidSelectorException with reason and errorCode for exception.

```
public InvalidSelectorException(java.lang.String reason,
                                java.lang.String errorCode)
```

Name	Description
reason	A description of the exception
errorCode	A string specifying the vendor specific error code.

The InvalidSelectorException Method

Constructs an InvalidSelectorException with reason. The error code defaults to null.

```
public InvalidSelectorException(java.lang.String reason)
```

Name	Description
reason	A description of the exception

7.2.8 class javax.jms.JMSSecurityException

```
public class JMSSecurityException
   extends JMSException
```

This exception is thrown when JMS IQ server rejects a user name/password submitted by a client. It is also thrown when any case of a security restriction prevents a method from completing.

The JMSSecurityException Method

Constructs a JMSSecurityException with reason.

```
public JMSSecurityExcpetion(java.lang.String reason,
                            java.lang.String errorCode)
```

Name	Description
reason	A description of the exception
errorCode	A string specifying the vendor specific error code.

The JMSSecurityException Method

Constructs a JMSSecurityException with reason. Error code default to null.

Name	Description
reason	A description of the exception

7.2.9 class javax.jms.MessageEOFException

```
public class MessageEOFException
   extends JMSException
```

This exception is thrown when an unexpected end of stream has been reached when a StreamMessage or BytesMessage is being read.

The MessageEOFException Method

Constructs a MessageEOFException with reason and errorCode for exception.

```
public MessageEOFExceptin(java.lang.String reason,
                          java.lang.String errorCode)
```

Name	Description
reason	A description of the exception
errorCode	A string specifying the vendor specific error code.

The MessageEOFException

Constructs a MessageEOFException with reason. The error code defauls to null.

```
public MessageEOFExcpetion(java.lang.String reason)
```

Name	Description
reason	A description of the exception

7.2.10 class javax.jms.MessageFormatException

```
public class MessageFormatException
   extends JMSException
```

This exception is thrown when a JMS client attempts to use a data type not supported by a message or attempts to read data in a message as the wrong type. It is also thrown when equivalent type errors are made with message property values. For example, this exception is thrown if `StreamMessage.writeObject` is given an unsupported class or if `StreamMessage.readShort` is used to read a `boolean` value. Note that the special case of a failure caused by an attempt to read improperly formatted `String` data as numeric values throw a java.lang.NumberFormatException.

The MessageFormatException Method

Constructs a MessageFormatException with reason and errorCode for exception.

```
public MessageFormatException(java.lang.String reason,
                              java.lang.String errorCode)
```

Name	Description
reason	A description of the exception
errorCode	A string specifying the vendor specific error code.

The MessageFormatExceptin

Constructs a MessageFormatExcecption with reason. The error code defauls to null.

```
public MessageFormatException(java.lang.String reason)
```

Name	Description
reason	A description of the exception

7.2.11 class javax.jms.MessageNotReadableException

```
public class MessageNotReadableException
   extends JMSException.
```

This exception is thrown when a JMS client attempts to read a write-only message.

The MessageNotReadableException Method

Constructs a MessageNotReadable with reason and errorCode for exception.

```
public MessageNotReadable(java.lang.String reason,
                          java.lang.String errorCode)
```

Name	Description
reason	A description of the exception
errorCode	A string specifying the vendor specific error code.

The MessageNotReadable Method

Constructs a MessageNotReadable with reason. The error code defauls to null.

```
public MessageNotReadable(java.lang.String reason)
```

Name	Description
reason	A description of the exception

7.2.12 class javax.jms.MessageNotWriteableException

```
public class MessageNotWriteableException
   extends JMSException
```

This exception is thrown when a JMS client attempts to write to a read-only message.

The MessageNotWriteableException Method

Constructs a MessageNotWriteableException with reason and errorCode for exception.

```
public MessageNotWriteableException(java.lang.String reason,
                          java.lang.String errorCode)
```

Name	Description
reason	A description of the exception
errorCode	A string specifying the vendor specific error code.

The MessageNotWriteableException Method

Constructs a MessageNotWriteableException with reason. The error code defauls to null.

```
public MessageNotWriteableException(java.lang.String reason)
```

Name	Description
reason	A description of the exception

7.2.13 class javax.jms.ResourceAllocationException

```
public class ResourceAllocationException
   extends JMSException
```

This exception is thrown when the JMS IQ server is unable to allocate the resources required by a method. For example, this exception is thrown when a call to

TopicConnectionFactory.createTopicConnection fails due to a lack of JMS provider resources.

The ResourceAllocationException Method

Constructs a ResourceAllocationException with reason and errorCode for exception.

```
public ResourceAllocationException(java.lang.String reason,
                                   java.lang.String errorCode)
```

Name	Description
reason	A description of the exception
errorCode	A string specifying the vendor specific error code.

The ResourceAllocationException Method

Constructs a ResourceAllocationException with reason. The error code defauls to null.

```
public ResourceAllocationException(java.lang.String reason)
```

Name	Description
reason	A description of the exception

7.2.14 class javax.jms.TransactionInProgressException

```
public class TransactionInProgressException
   extends JMSException
```

This exception is thrown when an operation is invalid because a transaction is in progress. For instance, an attempt to call Session.commit when a session is part of a distributed transaction will throw a TransactionInProgressException.

The TransactionInProgressException Method

Constructs a TransactionInProgressException with reason and errorCode for exception.

```
public TransactionInProgressException(java.lang.String reason,
                                   java.lang.String errorCode)
```

Name	Description
reason	A description of the exception
errorCode	A string specifying the vendor specific error code.

The TransactionInProgressException Method

Constructs a TransactionInProgressException with reason. The error code defauls to null.

```
public TransactionInProgressException(java.lang.String reason)
```

Name	Description
reason	A description of the exception

7.2.15 class javax.jms.TransactionRolledBackException

```
public class TransactionRolledBackException
   extends JMSException
```

This exception is thrown when a call to Session.commit results in a rollback of the current transaction.

The TransactionRolledBackException Method

Constructs a TransactionRolledBackExcxeption with reason and errorCode for exception.

```
public TransactionRolledBackException(java.lang.String reason,
                          java.lang.String errorCode)
```

Name	Description
reason	A description of the exception
errorCode	A string specifying the vendor specific error code.

The TransactionRolledBackException Method

Constructs a TransactionRolledBackException with reason. The error code defauls to null.

```
public TransactionRolledBackException(java.lang.String reason)
```

Name	Description
reason	A description of the exception

7.2.16 Unsupported JMS Classes

The current implementation of JMS Java APIs within the e*Gate API Kit DO NOT support the following classes:

- class javax.jms.QueueRequestor
- class javax.jms.TopicRequestor

7.3 Supported JMS Interfaces

The current implementation of JMS APIs within the e*Gate API Kit support the following interfaces:

7.3.1 interface javax.jms.Connection

```
public interface Connection
```

A Connection object is a client's active connection to the JMS IQ manager. It typically allocates JMS IQ manager resources outside the Java virtual machine (JVM).

A Connection serves several purposes:

- It encapsulates an open connection with a JMS provider. It typically represents an open TCP/IP socket between a client and a provider service daemon.
- Client authenticating takes place at it's creation.
- It can specify an unique client identifier.
- It provides ConnectionMetaData.
- It supports an optional ExceptionListener.

The close Method

Closes the connection.

```
close()
```

The getClientID Method

Gets the client identifier for this connection.

```
public java.lang.String getClientID()
   throwsJMSException
```

The getExceptionListener Method

Gets the `ExceptionListener` object for this connection.

```
public ExceptionListener getExceptionListener()
   throws JMSException
```

The getMetaData Method

Gets the metadata for this connection.

```
public ConnectionMetaData getMetaData()
   throws JMSException
```

The setClientID Method

Sets the client identifier for this connection.

```
public void setClientID(java.lang.String clientID)
   throws JMSException
```

Name	Description
clientID	The unique client identifier.

The setExceptionListener Method

Sets an exception listener for this connection.

```
public void setExceptionListener(ExceptionListener listener)
   throws JMSException
```

Name	Description
listener	The exception listener.

The Start Method

Starts (or restarts) a connection's delivery of incoming messages. A call to `start` a connection that has already been started is ignored.

```
public void start()
   throws JMSException
```

The Stop Method

Temporarily stops a connection's delivery of incoming messages. Delivery can be restarted using the connection's `start` method.

```
public void stop()
   throws JMSException
```

7.3.2 interface javax.jms.QueueConnection

```
public interface QueueConnection
extends Connection
```

A QueueConnection is an active connection to a JMS PTP provider. A client uses a QueueConnection to create one or more QueueSessions for producing and consuming messages.

The createQueueSession Method

Creates a QueueSession.

```
public QueueSession createQueueSession(boolean transacted,int
acknowledgeMode)
   throws JMSException
```

Name	Description
transacted	If true, the session is transacted.
acknowledeMode	Indicates whether the consumer or the client will acknowledge any any messages that it receives. This parameter is ignored if the session is transacted. Legal valuees are: `Session.AUTO_ACKNOWLEDGE`, `Session.CLIENT_ACKNOWLEDGE` and `Session.DUPS_OK_ACKNOWLEDGE`.

Throws JMSException if JMS Connection fails to create a session due to some internal error or lack of support for specific transaction and acknowledgement mode.

7.3.3 interface javax.jms.XAQueueConnection

```
public interface XAQueueConnection
extends XAConnection, QueueConnection
```

XAQueueConnection provides the same create options as QueueConnection (optional). The only difference is that an XAConnection is by definition transacted.

createXAQueueSession

```
public XAQueueSession createXAQueueSession()
   throws JMSException
```

Create an XAQueueSession.

Throws JMSException if JMS Connection fails to create a XA queue session due to some internal error.

createQueueSession

```
public QueueSession createQueueSession(boolean transacted, int
acknowledgeMode)
   throws JMSException
```

Create an XAQueueSession.

Specified by

createQueueSession in interface QueueConnection

Name	Description
transacted	ignored
acknowledgeMode	ignored.

Throws JMSException if JMS Connection fails to create a XA queue session due to some internal error.

7.3.4 interface javax.jms.TopicConnection

```
public interface TopicConnection
   extends Connection
```

A TopicConnection object is an active connection to the JMS IQ server, used in the Pub/Sub mode. A client uses a TopicConnection object to create one or more TopicSession objects for producing and consuming messages.

The createTopicSession Method

Creates a `TopicSession` object.

```
public TopicSession createTopicSession(boolean transacted, int
acknowledgeMode)
   throws JMSException
```

Name	Description
transacted	Indicates whether the session is transacted.
acknowledgeMode	Indicates whether the consumer or the client will acknowledge any messages it receives; ignored if the session is transacted. Legal values are `Session.AUTO_ACKNOWLEDGE`, `Session.CLIENT_ACKNOWLEDGE`, and `Session.DUPS_OK_ACKNOWLEDGE`.

Throws JMSException if JMS Connection fails to create a session due to some internal error or lack of support for specific transaction and acknowledgement mode.

7.3.5 interface javax.jms.XATopicConnection

```
public interface XATopicConnection
   extends XAConnection, TopicConneciton
```

An XATopicConnection provides the same create options as TopicConnection (optional). The only difference is that an XAConnection is by definition transacted.

The createTopicSession Method

Creates an XATopicSession object.

```
public TopicSession createTopicSession(boolean transacted, int
acknowledgeMode)
   throws JMSException
```

Name	Description
transacted	Indicates whether the session is transacted. Ignored
acknowledgeMode	Indicates whether the consumer or the client will acknowledge any messages it receives; ignored if the session is transacted. Legal values are Session.AUTO_ACKNOWLEDGE, Session.CLIENT_ACKNOWLEDGE, and Session.DUPS_OK_ACKNOWLEDGE. Ignored

Throws JMSException if JMS Connection fails to create a XA topic session due to some internal error.

createXATopicSession

Creates an XATopicSession.

```
public XATopicSession createXATopicSession()
   throws JMSException
```

Throws JMSException if JMS Connection fails to create a XA topic session due to some internal error.

Throws JMSException if JMS Connection fails to create a XA topic session due to some internal error.

7.3.6 interface javax.jms.ConnectionFactory

```
public interface ConnectionFactory
```

A ConnectionFactory encapsulates a set of connection configuration parameters that has been defined by an administrator. A client uses it to create a Connection with a JMS provider. ConnectionFactory objects support concurrent use. A ConnectionFactory is a JMS administered object.

JMS administered objects are objects containing JMS configuration information that are created by a JMS administrator and later used by JMS clients. They make it practical to administer JMS in the enterprise. Although the interfaces for administered objects do not explicitly depend on JNDI, JMS establishes the convention that JMS clients find them by looking them up in a JNDI namespace.

An administrator can place an administered object anywhere in a namespace. JMS does not define a naming policy. It is expected that JMS providers will provide the tools that the administrator needs to create and configure administered objects in a JNDI namespace. JMS provider implementations of administered objects should be both `javax.jndi.Referenceable` and `java.io.Serializable` so that they can be stored in all JNDI naming contexts. In addition, it is recommended that these implementations follow the JavaBeans(TM) design patterns.

This strategy provides several benefits:

- It hides provider-specific details from JMS clients.
- It abstracts JMS administrative information into Java objects that are easily organized and administrated from a common management console.
- Since there will be JNDI providers for all popular naming services, this means JMS providers can deliver one implementation of administered objects that will run everywhere.

An administered object should not hold on to any remote resources. Its lookup should not use remote resources other than those used by JNDI itself. Clients should think of administered objects as local Java objects. Looking them up should not have any hidden side affects or use surprising amounts of local resources.

7.3.7 interface javax.jms.QueueConnectionFactory

```
public interface QueueConnectionFactory
extends ConnectionFactory
```

A client uses a QueueConnectionFactory to create QueueConnections with a JMS PTP provider.

The createQueueConnection Method

```
public QueueConnection createQueueConnection()
   throws JMSException
```

Create a queue connection with default user identity. The connection is created in stopped mode. No messages will be delivered until Connection.start method is explicitly called.

Throws JMSException if JMS Provider fails to create Queue Connection due to some internal error. required resources for a Queue Connection.

Throws JMSSecurityException if client authentication fails due to invalid user name or password.

The createQueueConnection Method

```
public QueueConnection createQueueConnection(java.lang.String
userName, java.lang.String password)
   throws JMSException
```

Create a queue connection with specified user identity. The connection is created in stopped mode. No messages will be delivered until Connection.start method is explicitly called.

Name	Description
userName	The caller's user name.
password	The caller's password.

Throws JMSException if JMS Provider fails to create Queue Connection due to some internal error.

Throws JMSSecurityException if client authentication fails due to invalid user name or password.

7.3.8 interface javax.jms.XAConnectionFactory

```
public interface XAConnectionFactory
```

To include JMS transactions in a JTS transaction, an application server requires a JTS aware JMS provider. A JMS provider exposes its JTS support using a JMS XAConnectionFactory which an application server uses to create XASessions.

XAConnectionFactory's are JMS administered objects just like ConnectionFactory objects. It is expected that application servers will find them using JNDI.

7.3.9 interface javax.jms.TopicConnectionFactory

```
public interface TopicConnectionFactory
   extends ConnectionFactory
```

A client uses a TopicConnectionFactory object to create TopicConnection objects with the JMS IQ manager, while implementing Pub/Sub mode.

The createTopicConnection Method

Creates a topic connection with the default user identity. The connection is created in stopped mode. No messages will be delivered until the `Connection.start` method is explicitly called.

```
public TopicConnection createTopicConnection()
   throws JMSException
```

Throws JMSException if JMS Provider fails to create a Topic Connection due to some internal error.

Throws JMSSecurityException if client authentication fails due to invalid user name or password.

The createTopicConnection Method

Creates a topic connection with the specified user identity. The connection is created in stopped mode. No messages will be delivered until the Connection.start method is explicitly called.

```
public TopicConnection createTopicConnection(java.lang.String
userName, java.lang.String password)
   throws JMSException
```

Name	Description
userName	The caller's user name.
password	The caller's password

Throws JMSException if JMS Provider fails to create a Topic Connection due to some internal error.

Throws JMSSecurityException if client authentication fails due to invalid user name or password.

7.3.10 interface javax.jms.XATopicConnectionFactory

```
public interface XATopicConnectionFactory
   extends XAConnectionFactory, TopicConnecitonFactory
```

An XATopicConnectionFactory provides the same create options as a TopicConnectionFactory (optional).

The createXATopicConnection Method

Creates a XA topic connection with the default user identity. The connection is created in stopped mode. No messages will be delivered until the `Connection.start` method is explicitly called.

```
public XATopicConnection createXATopicConnection()
   throws JMSException
```

Throws JMSException if JMS Provider fails to create XA topic Connection due to some internal error.

Throws JMSSecurityException if client authentication fails due to invalid user name or password.

The createXATopicConneciton Method

Creates a XA topic connection with the specified user identity. The connection is created in stopped mode. No messages will be delivered until the Connection.start method is explicitly called.

```
public XATopicConnection createXATopicConnection(java.lang.String
userName, java.lang.String password)
   throws JMSException
```

Name	Description
userName	The caller's user name.
password	The caller's password

Throws JMSException if JMS Provider fails to create XA topi connection due to some internal error.

Throws JMSSecurityException if client authentication fails due to invalid user name or password.

7.3.11 interface javax.jms.ConnectionMetaData

```
public interface ConnectionMetaData
```

A ConnectionMetaData object provides information describing the Connection object.

The getJMSVersion Method

```
public java.lang.String getJMSVersion()
   throws JMSException
```

Get the JMS version.

Throws JMSException if some internal error occurs in JMS implementation during the meta-data retrieval.

The getJMSMajorVersion Method

Gets the JMS major version number.

```
public int getJMSMajorVersion()
   throws JMSException
```

Throws JMSException if some internal error occurs in JMS implementation during the meta-data retrieval.

The getJMSMinorVersion Method

Gets the JMS minor version number.

```
public int getJMSMinorVersion()
   throws JMSException
```

Throws JMSException if some internal error occurs in JMS implementation during the meta-data retrieval.

The getJMSProviderName Method

Gets the JMS provider name.

```
public java.lang.String getJMSProviderName()
   throws JMSException
```

Throws JMSException if some internal error occurs in JMS implementation during the meta-data retrieval.

The getProviderVersion Method

Gets the JMS provider version.

```
public java.lang.String getProviderVersion()
   throws JMSException
```

Throws JMSException if some internal error occurs in JMS implementation during the meta-data retrieval.

The getProviderMajorVersion Method

Gets the JMS provider major version number.

```
public int getProviderMajorVersion()
   throws JMSException
```

Throws JMSException if some internal error occurs in JMS implementation during the meta-data retrieval.

The getProviderMinorVersion Method

Gets the JMS provider minor version number.

```
public int getProviderMinorVersion()
   throws JMSException
```

Throws JMSException if some internal error occurs in JMS implementation during the meta-data retrieval.

The getJMSXPropertyNames Method

Gets an enumeration of the JMSX property names.

```
public java.util.Enumeration getJMSXPropertyNames()
   throws JMSException
```

Throws JMSException if some internal error occurs in JMS implementation during the property names retrieval.

7.3.12 interface javax.jms.DeliveryMode

```
public interface DeliveryMode
```

The delivery modes supported by the JMS API are PERSISTENT and NON_PERSISTENT.

A client marks a message as persistent if it feels that the application will have problems if the message is lost in transit. A client marks a message as non-persistent if an occasional lost message is tolerable. Clients use delivery mode to tell the JMS IQ manager how to balance message transport reliability throughput.

Delivery mode only covers the trasport of the message to its destination. Retention of a message at the destination until its receipt is acknowledged is not guaranteed by a PERSISTENT delivery mode. Clients should assume that message retention policies are set administratively. Message retention policy governs the reliability of message delivery from destination to message consumer. For example, if a client's message storage space is exhausted, some messages as defined by a site specific message retention policy may be dropped.

A message is guaranteed to be delivered once-and-only-once by a JMS Provider if the delivery mode of the message is persistent and if the destination has a sufficient message retention policy.

NON_PERSISTENT Field

This is the lowest overhead delivery mode because it does not require that the message be logged to stable storage. The level of JMS provider failure that causes a NON_PERSISTENT message to be lost is not defined.

A JMS provider must deliver a NON_PERSISTENT message with an at-most-once guarantee. This means it may lose the message but it must not deliver it twice.

```
public static final int NON_PERSISTENT
```

PERSISTENT Field

This mode instructs the JMS provider to log the message to stable storage as part of the client's send operation. Only a hard media failure should cause a PERSISTENT message to be lost.

7.3.13 interface javax.jms.Destination

```
public interface Destination
```

A Destination object encapsulates a JMS IQ manager-specific address. public interface.

7.3.14 interface javax.jms.Queue

```
public interface Queue
   extends Destination
```

A Queue object encapsulates a provider-specific queue name. In this manner, a client specifies the identity of queue to JMS methods. The actual length of time messages are held by a queue and the consequences of resource overflow are not defined by JMS.

The getQueueName Method

```
public java.lang.String getQueueName()
```

```
    throws JMSException
```

Get the name of this queue. Clients that depend upon the name, are not portable.

Throws JMSException if JMS implementation for Queue fails to to return queue name due to some internal error.

The toString Method

```
public java.lang.String toString()
```

Return a pretty printed version of the queue name

Overrides:

toString in class java.lang.Object

7.3.15 interface javax.jms.TemporaryQueue

```
public interface TemporaryQueue
    extends Queue
```

A TemporaryQueue is a unique Queue object created for the duration of a QueueConnection. It is a system defined queue that can only be consumed by the QueueConnection that created it.

The delete Method

```
public void delete()
    throws JMSException
```

Delete this temporary queue. If there are still existing senders or receivers still using it, then a JMSException will be thrown.

Throws JMSException if JMS implementation fails to delete a Temporary topic due to some internal error.

7.3.16 interface javax.jms.Topic

```
public interface Topic
    extends Destination
```

A Topic object encapsulates a provider-specific topic name. The topic object provides the means for a client to specify the identity of a topic to JMS methods.

Many Pub/Sub implementations group topics into hierarchies and provide various options for subscribing to parts of the hierarchy. JMS places no restriction on what a Topic object represents.

The getTopicName Method

Gets the name of this topic.

```
public java.lang.String getTopicName()
    throws JMSException
```

Throws JMSException if JMS implementation for Topic fails to to return topic name due to some internal error.

The toString Method

Returns a string representation of this object.

```
public java.lang.String toString()
```

Overrides:

`toString` in class `java.lang.Object`

7.3.17 interface javax.jms.TemporaryTopic

```
public interface TemporaryTopic
   extends Topic
```

A TemporaryTopic object is a unique Topic object created for the duration of a TopicConnection. It is a system-defined topic that can be consumed only by the TopicConnection that created it.

The delete Method

Deletes this temporary topic. If there are existing subscribers still using it, a `JMSException` will be thrown.

```
public void delete()
   throws JMSException
```

Throws JMSException if JMS implementation fails to delete a Temporary queue due to some internal error.

7.3.18 interface javax.jms.ExceptionListener

```
public interface ExceptionListener
```

If the JMS IQ manager detects a serious problem with a Connection object, it informs the Connection object's ExceptionListener, if one has been registered. It does this by calling the listener's onException method, passing it a JMSException argument describing the problem.

This allows a client to be asynchronously notified of a problem. Some Connections only consume messages so they would have no other way to learn their Connection has failed.

A JMS provider should attempt to resolve connection problems themselves prior to notifying the client of them.

The onException Method

Notifies user of a JMS exception.

```
public void onException(JMSException exception)
```

Name	Description
exception	The JMS exception.

7.3.19 interface javax.jms.Message

```
public interface Message
   getJMSMessageID
```

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

JMS Messages are composed of the following parts:

- Header - All messages support the same set of header fields. Header fields contain values used by both clients and providers to identify and route messages.
- Properties - Each message contains a built-in facility for supporting application defined property values. Properties provide an efficient mechanism for supporting application defined message filtering.
- Body - JMS defines several types of message body which cover the majority of messaging styles currently in use.

JMS defines five types of message body:

- Stream - a stream of Java primitive values. It is filled and read sequentially.
- Map - a set of name-value pairs where names are Strings and values are Java primitive types. The entries can be accessed sequentially or randomly by name. The order of the entries is undefined.
- Text - a message containing a java.util.String. The inclusion of this message type is based on our presumption that XML will likely become a popular mechanism for representing content of all kinds including the content of JMS messages.
- Object - a message that contains a Serializable java object
- Bytes - a stream of uninterpreted bytes. This message type is for literally encoding a body to match an existing message format. In many cases, it will be possible to use one of the other, easier to use, body types instead. Although JMS allows the use of message properties with byte messages it is typically not done since the inclusion of properties may affect the format.

The JMSCorrelationID header field is used for linking one message with another. It typically links a reply message with its requesting message. JMSCorrelationID can hold either a provider-specific message ID, an application-specific String or a provider-native byte[] value.

A Message contains a built-in facility for supporting application defined property values. In effect, this provides a mechanism for adding application specific header fields to a message. Properties allow an application, via message selectors, to have a JMS provider select/filter messages on its behalf using application-specific criteria. Property names must obey the rules for a message selector identifier. Property values can be boolean, byte, short, int, long, float, double, and String.

Property values are set prior to sending a message. When a client receives a message, its properties are in read-only mode. If a client attempts to set properties at this point, a MessageNotWriteableException is thrown. If clearProperties is called, the properties can now be both read from and written to. Note that header fields are distinct from properties. Header fields are never in a read-only mode.

A property value may duplicate a value in a message's body or it may not. Although JMS does not define a policy for what should or should not be made a property, application developers should note that JMS providers will likely handle data in a message's body more efficiently than data in a message's properties. For best performance, applications should only use message properties when they need to customize a message's header. The primary reason for doing this is to support customized message selection.

In addition to the type-specific set/get methods for properties, JMS provides the setObjectProperty and getObjectProperty methods. These support the same set of property types using the objectified primitive values. Their purpose is to allow the decision of property type to made at execution time rather than at compile time. They support the same property value conversions.

The setObjectProperty method accepts values of class Boolean, Byte, Short, Integer, Long, Float, Double and String. An attempt to use any other class must throw a JMSException.

The getObjectProperty method only returns values of class Boolean, Byte, Short, Integer, Long, Float, Double and String.

The order of property values is not defined. To iterate through a message's property values, use getPropertyNames to retrieve a property name enumeration and then use the various property get methods to retrieve their values.

A message's properties are deleted by the clearProperties method. This leaves the message with an empty set of properties.

Getting a property value for a name which has not been set returns a null value. Only the getStringProperty and getObjectProperty methods can return a null value. The other property get methods must throw a java.lang.NullPointerException if they are used to get a non-existent property.

JMS reserves the `JMSX' property name prefix for JMS defined properties. The full set of these properties is defined in the Java Message Service specification. New JMS defined properties may be added in later versions of JMS. Support for these properties is optional. The String[] ConnectionMetaData.getJMSXPropertyNames method returns the names of the JMSX properties supported by a connection.

JMSX properties may be referenced in message selectors whether or not they are supported by a connection. If they are not present in a message, they are treated like any other absent property. property.

JSMX properties `set by provider on send' are available to both the producer and the consumers of the message. JSMX properties `set by provider on receive' are only available to the consumers.

JMSXGroupID and JMSXGroupSeq are simply standard properties clients should use if they want to group messages. All providers must support them. Unless specifically noted, the values and semantics of the JMSX properties are undefined.

JMS reserves the `JMS_' property name prefix for provider-specific properties. Each provider defines there own value of . This is the mechanism a JMS provider uses to make its special per message services available to a JMS client.

The purpose of provider-specific properties is to provide special features needed to support JMS use with provider-native clients. They should not be used for JMS to JMS messaging.

JMS provides a set of message interfaces that define the JMS message model. It does not provide implementations of these interfaces.

Each JMS provider supplies a set of message factories with its Session object for creating instances of these messages. This allows a provider to use implementations tailored to their specific needs.

A provider must be prepared to accept message implementations that are not its own. They may not be handled as efficiently as their own implementations; however, they must be handled.

A JMS message selector allows a client to specify by message header the messages it's interested in. Only messages whose headers match the selector are delivered. The semantics of not delivered differ a bit depending on the MessageConsumer being used (see QueueReceiver and TopicSubscriber).

Message selectors cannot reference message body values.

A message selector matches a message when the selector evaluates to true when the message's header field and property values are substituted for their corresponding identifiers in the selector.

A message selector is a String, whose syntax is based on a subset of the SQL92 conditional expression syntax.

The order of evaluation of a message selector is from left to right within precedence level. Parenthesis can be used to change this order.

Predefined selector literals and operator names are written here in upper case; however, they are case insensitive.

A selector can contain:

- Literals:
  - A string literal is enclosed in single quotes with single quote represented by doubled single quote such as `literal' and `literal''s'; like Java string literals these use the unicode character encoding.
  - An exact numeric literal is a numeric value without a decimal point such as 57, -957, +62; numbers in the range of Java long are supported. Exact numeric literals use the Java integer literal syntax.
  - An approximate numeric literal is a numeric value in scientific notation such as 7E3, -57.9E2 or a numeric value with a decimal such as 7., -95.7, +6.2; numbers in the range of Java double are supported. Approximate literals use the Java floating point literal syntax.
  - The boolean literals TRUE, true, FALSE and false.
- Identifiers:
  - An identifier is an unlimited length sequence of Java letters and Java digits, the first of which must be a Java letter. A letter is any character for which the method Character.isJavaLetter returns true. This includes `_' and `$'. A letter or digit is any character for which the method Character.isJavaLetterOrDigit returns true.
  - Identifiers cannot be the names NULL, TRUE, or FALSE.
  - Identifiers cannot be NOT, AND, OR, BETWEEN, LIKE, IN, and IS.
  - Identifiers are either header field references or property references.
  - Identifiers are case sensitive.

- Message header field references are restricted to JMSDeliveryMode, MSPriority, JMSMessageID, JMSTimestamp, JMSCorrelationID, and JMSType. JMSMessageID, JMSCorrelationID, and JMSType values may be null and if so are treated as a NULL value.
- Any name beginning with `JMSX' is a JMS defined property name.
- Any name beginning with `JMS_' is a provider-specific property name.
- Any name that does not begin with `JMS' is an application-specific property name. If a property is referenced that does not exist in a message its value is NULL. If it does exist, its value is the corresponding property value.

- Whitespace is the same as that defined for Java: space, horizontal tab, form feed and line terminator.
- Expressions:
  - A selector is a conditional expression; a selector that evaluates to true matches; a selector that evaluates to false or unknown does not match.
  - Arithmetic expressions are composed of themselves, arithmetic operations, identifiers (whose value is treated as a numeric literal) and numeric literals.
  - Conditional expressions are composed of themselves, comparison operations and logical operations.
- Standard bracketing () for ordering expression evaluation is supported.
- Logical operators in precedence order: NOT, AND, OR
- Comparison operators: =, >, >=, <, <=, <> (not equal)
  - Only like type values can be compared. One exception is that it is valid to compare exact numeric values and approximate numeric values (the type conversion required is defined by the rules of Java numeric promotion). If the comparison of non-like type values is attempted, the selector is always false.
  - String and boolean comparison is restricted to = and <>. Two strings are equal if and only if they contain the same sequence of characters.
- Arithmetic operators in precedence order:
  - +, - unary
  - *, / multiplication and division
  - +, - addition and subtraction
  - Arithmetic operations on a NULL value are not supported; if they are attempted, the complete selector is always false.
  - Arithmetic operations must use Java numeric promotion.
- arithmetic-expr1 [NOT] BETWEEN arithmetic-expr2 and arithmetic-expr3 comparison operator
  - age BETWEEN 15 and 19 is equivalent to age >= 15 AND age <= 19
  - age NOT BETWEEN 15 and 19 is equivalent to age < 15 OR age > 19

   - If any of the exprs of a BETWEEN operation are NULL the value of the operation is false; if any of the exprs of a NOT BETWEEN operation are NULL the value of the operation is true.

- identifier [NOT] IN (string-literal1, string-literal2,...) comparison operator where identifer has a String or NULL value.
   - Country IN (' UK', 'US', 'France') is true for `UK' and false for `Peru' it is equivalent to the expression (Country = ' UK') OR (Country = ' US') OR (Country = ' France')
   - Country NOT IN (' UK', 'US', 'France') is false for `UK' and true for `Peru' it is equivalent to the expression NOT ((Country = ' UK') OR (Country = ' US') OR (Country = ' France'))
   - If identifier of an IN or NOT IN operation is NULL the value of the operation is unknown.
- identifier [NOT] LIKE pattern-value [ESCAPE escape-character] comparison operator, where identifier has a String value; pattern-value is a string literal where `_' stands for any single character; `%' stands for any sequence of characters (including the empty sequence); and all other characters stand for themselves. The optional escape-character is a single character string literal whose character is used to escape the special meaning of the `_' and `%' in pattern-value.
   - phone LIKE `12%3' is true for `123' `12993' and false for `1234'
   - word LIKE `l_se' is true for `lose' and false for `loose'
   - underscored LIKE `\_%' ESCAPE `\' is true for `_foo' and false for `bar'
   - phone NOT LIKE `12%3' is false for `123' `12993' and true for `1234'
   - If identifier of a LIKE or NOT LIKE operation is NULL the value of the operation is unknown.
- identifier IS NULL comparison operator tests for a null header field value, or a missing property value.
   - prop_name IS NULL
- identifier IS NOT NULL comparison operator tests for the existence of a non null header field value or a property value.
   - prop_name IS NOT NULL

JMS providers are required to verify the syntactic correctness of a message selector at the time it is presented. A method providing a syntactically incorrect selector must result in a JMSException.

The following message selector selects messages with a message type of car and color of blue and weight greater than 2500 lbs:

```
"JMSType = `car' AND color = `blue' AND weight > 2500"
```

As noted above, property values may be NULL. The evaluation of selector expressions containing NULL values is defined by SQL 92 NULL semantics. A brief description of these semantics is provided here.

SQL treats a NULL value as unknown. Comparison or arithmetic with an unknown value always yields an unknown value.

The IS NULL and IS NOT NULL operators convert an unknown value into the respective TRUE and FALSE values.

When used in a message selector JMSDeliveryMode is treated as having the values `PERSISTENT' and `NON_PERSISTENT'.

Although SQL supports fixed decimal comparison and arithmetic, JMS message selectors do not. This is the reason for restricting exact numeric literals to those without a decimal (and the addition of numerics with a decimal as an alternate representation for an approximate numeric values). SQL comments are not supported.

DEFAULT_DELIVERY_MODE

The message producer's default delivery mode is persistent.

```
public static final int DEFAULT_DELIVERY_MODE
```

DEFAULT_PRIORITY

The message producer's default priority is 4.

```
public static final int DEFAULT_PRIORITY
```

DEFAULT_TIME_TO_LIVE

The message producer's default time to live is unlimited, the message never expires.

```
public static final long DEFAULT_TIME_TO_LIVE
```

The getJMSMessageID Method

Gets the message ID. The messageID header field contains a value that uniquely identifies each message sent by a provider. When a message is sent, messageID can be ignored. When the send method returns it contains a provider-assigned value.

A JMSMessageID is a String value which should function as a unique key for identifying messages in a historical repository. The exact scope of uniqueness is provider defined. It should at least cover all messages for a specific installation of a provider where an installation is some connected set of message routers.

All JMSMessageID values must start with the prefix `ID:'. Uniqueness of message ID values across different providers is not required.

Since message IDs take some effort to create and increase the message size, some JMS providers may be able to optimize message overhead if they are given a hint that message ID is not used by an application. JMS message Producers provide a hint to disable message ID. When a client sets a Producer to disable message ID they are saying that they do not depend on the value of message ID for the messages it produces. These messages must either have message ID set to null or, if the hint is ignored, messageID must be set to its normal unique value.

```
public java.lang.String getJMSMessageID()
   throws JMSException
```

Throws JMSException if JMS fails to get the message Id due to internal JMS error.

The setJMSMessageID Method

Sets this message's ID. Providers set this field when a message is sent. This operation can be used to change the value of a message that's been received.

```
public void setJMSMessageID(java.lang.String id)
   throws JMSException
```

Name	Description
id	The identifier of the message.

Throws JMSException if JMS fails to set the message Id due to internal JMS error.

The getJMSTimestamp Method

Gets this message's timestamp. The JMSTimestamp header field contains the time a message was handed off to a provider to be sent. It is not the time the message was actually transmitted because the actual send may occur later due to transactions or other client side queueing of messages.

```
public long getJMSTimestamp()
   throws JMSException
```

Throws JMSException if JMS fails to get the Timestamp due to internal JMS error.

The setJMSTimestamp Method

Sets this message's timestamp.

```
public void setJMSTimestamp(long timestamp)
   throws JMSException
```

Name	Description
long timestamp	Returns the messages timestamp.

Throws JMSException if JMS fails to set the Timestamp due to internal JMS error.

The getJMSCorrelationIDAsBytes Method

Gets the correlation ID as an array of bytes for this message.

```
public byte[] getJMSCorrelationIDAsBytes()
   throws JMSException
```

Throws JMSException if JMS fails to get correlationId due to some internal JMS error.

The setJMSCorrelationIDAsBytes Method

Sets the correlation ID as an array of bytes for this message. The array is copied before the method returns, so future modifications to the array will not alter this message header.

```
public void setJMSCorrelationIDAsBytes(byte[] correlationID)
   throws JMSException
```

Name	Description
correlationID	The correlation identifier value as an array of bytes.

Throws JMSException if JMS fails to set correlationId due to some internal JMS error.

The setJMSCorrelationID Method

Sets the correlation ID for the message.

```
public void setJMSCorrelationID(java.lang.String correlationID)
   throws JMSException
```

JMSCorrelationID can hold one of the following:

- A provider-specific message ID
- An application-specific String
- A provider-native byte[] value

Since each message sent by the JMS IQ manager is assigned a message ID value, it is convenient to link messages via message ID. All message ID values must start with the 'ID:' prefix.

Name	Description
correlationID	The message identifier of the message being referred to.

Throws JMSException if JMS fails to set correlationId due to some internal JMS error.

The getJMSCorrelationID Method

Gets the correlation ID for the message.

```
public java.lang.String getJMSCorrelationID()
   throws JMSException
```

Throws JMSException if JMS fails to get correlationId due to some internal JMS error.

The getJMSReplyTo Method

Gets the Destination object to which a reply to this message should be sent.

```
public Destination getJMSReplyTo()
   throws JMSException
```

Throws JMSException if JMS fails to get ReplyTo Destination due to some internal JMS error.

The setJMSReplyTo Method

Sets the Destination object to which a reply to this message should be sent. The replyTo header field contains the destination where a reply to the current message should be sent. If it is null no reply is expected. The destination may be either a Queue or a Topic.

```
public void setJMSReplyTo(Destination replyTo)
   throws JMSException
```

Name	Description
replyTo	The `destination` to send the response to for this message.

Throws JMSException if JMS fails to set ReplyTo Destination due to some internal JMS error.

The getJMSDestination Method

Gets the Destination object for this message. The destination field contains the destination to which the message is being sent. When a message is sent this value is ignored. After completion of the send method it holds the destination specified by the send. When a message is received, its destination value must be equivalent to the value assigned when it was sent.

```
public Destination getJMSDestination()
   throws JMSException
```

Throws JMSException if JMS fails to get JMS Destination due to some internal JMS error.

The setJMSDestination Method

The JMS IQ manager sets this field when a message is sent. This method can be used to change the value for a message that has been received.

```
public void setJMSDestination(Destination destination)
   throws JMSException
```

Name	Description
destination	The `destination` for this message.

Throws JMSException if JMS fails to set JMS Destination due to some internal JMS error.

The getJMSDeliveryMode Method

Gets the DeliveryMode value specified for this message.

```
public int getJMSDeliveryMode()
   throws JMSException
```

Throws JMSException if JMS fails to get JMS DeliveryMode due to some internal JMS error.

The setJMSDeliveryMode Method

Sets the DeliveryMode value for this message.

```
public void setJMSDeliveryMode(int deliveryMode)
   throws JMSException
```

Name	Description
deliveryMode	The delivery mode for this message.

Throws JMSException if JMS fails to set JMS DeliveryMode due to some internal JMS error.

The getJMSRedelivered Method

Gets an indication of whether this message is being redelivered. If a client receives a message with the redelivered indicator set, it is likely, but not guaranteed, that this message was delivered to the client earlier but the client did not acknowledge its receipt at that earlier time.

```
public boolean getJMSRedelivered()
```

```
    throws JMSException
```

Throws JMSException if JMS fails to get JMS Redelivered flag due to some internal JMS error.

The setJMSRedelivered Method

Specifies whether this message is being redelivered. This field is set at the time the message is delivered. This operation can be used to change the value of a message that's been received.

```
public void setJMSRedelivered(boolean redelivered)
    throws JMSException
```

Name	Description
redelivered	An indication of whether this message is being redelivered.

Throws JMSException if JMS fails to set JMS Redelivered flag due to some internal JMS error.

The getJMSType Method

Gets the message type identifier supplied by the client when the message was sent.

```
public java.lang.String getJMSType()
    throws JMSException
```

The setJMSType Method

Sets the message type.

```
public void setJMSType(java.lang.String type)
    throws JMSException
```

Name	Description
type	The message type.

Throws JMSException if JMS fails to get JMS message type due to some internal JMS error.

The setJMSType Method

Set the message type.

```
public void setJMSType(java.lang.String type)
    throws JMSException
```

Name	Description
type	The class of the message.

Throws JMSException if JMS fails to set JMS message type due to some internal JMS error.

The getJMSExpiration Method

Gets the message's expiration value. When a message is sent, expiration is left unassigned. After completion of the send method, it holds the expiration time of the message. This is the sum of the time-to-live value specified by the client and the GMT at the time of the send.

If the time-to-live is specified as zero, expiration is set to zero which indicates the message does not expire. When a message's expiration time is reached, a provider should discard it. JMS does not define any form of notification of message expiration.

```
public long getJMSExpiration()
   throws JMSException
```

Throws JMSException if JMS fails to get JMS message expiration due to some internal JMS error.

The setJMSExpiration Method

Sets the message's expiration value.

```
public void setJMSExpiration(long expiration)
   throws JMSException
```

Name	Description
expiration	The messages expiration time.

Throws JMSException if JMS fails to set JMS message expiration due to some internal JMS error.

The getJMSPriority Method

Gets the message's priority value. JMS defines a ten level priority value with 0 as the lowest priority and 9 as the highest. In addition, clients should consider priorities 0-4 as gradations of normal priority and priorities 5-9 as gradations of expedited priority.

```
public long getJMSPriority()
   throws JMSException
```

Throws JMSException if JMS fails to get JMS message priority due to some internal JMS error.

The setJMSPriority Method

Sets the priority level for this message.

```
public void setJMSPriority(int priority)
   throws JMSException
```

Name	Description
priority	The default priority of this message.

Throws JMSException if JMS fails to set JMS message priority due to some internal JMS error.

The clearProperties Method

Clears a message's properties. The message header fields and body are not cleared.

```
public void clearProperties()
   throws JMSException
```

Throws JMSException if JMS fails to clear JMS message properties due to some internal JMS error.

The propertyExists Method

Queries whether a property value exists.

```
public boolean propertyExists(java.lang.String name)
   throws JMSException
```

Name	Description
name	The name of the property to test.

Throws JMSException if JMS fails to check if property exists due to some internal JMS error.

The getBooleanProperty Method

Returns the value of the boolean property with the specified name.

```
public boolean getBooleanProperty(java.lang.String name)
   throws JMSException
```

Name	Description
name	The name of the boolean property.

Throws JMSException if JMS fails to get Property due to some internal JMS error.

Throws MessageFormatException - if this type conversion is invalid.

The getByteProperty Method

Returns the value of the byte property with the specified name.

```
public byte getByteProperty(java.lang.String name)
   throws JMSException
```

Name	Description
name	The name of the byte property.

Throws JMSException if JMS fails to get Property due to some internal JMS error.

Throws MessageFormatException - if this type conversion is invalid.

The getShortProperty Method

Returns the value of the short property with the specified name.

```
public short getShortProperty(java.lang.String name)
   throws JMSException
```

Name	Description
name	The name of the short property.

Throws JMSException if JMS fails to get Property due to some internal JMS error.

Throws MessageFormatException if this type conversion is invalid.

The getIntProperty Method

Returns the value of the int property with the specified name.

```
public int getIntProperty(java.lang.String name)
   throws JMSException
```

Name	Description
name	The name of the int property.

Throws JMSException if JMS fails to get Property due to some internal JMS error.

Throw MessageFormatException if this type conversion is invalid.

The getLongProperty Method

Returns the value of the long property with the specified name.

```
public long getLongProperty(java.lang.String name)
   throws JMSException
```

Name	Description
name	The name of the long property.

Throws JMSException if JMS fails to get Property due to some internal JMS error.

Throws MessageFormatException- if this type conversion is invalid.

The getFloatProperty Method

Returns the value of the float property with the specified name.

```
public float getFloatProperty(java.lang.String name)
   throws JMSException
```

Name	Description
name	The name of the float property.

Throws JMSException if JMS fails to get Property due to some internal JMS error.

Throw MessageFormatException if this type conversion is invalid.

The getDoubleProperty Method

Returns the value of the double property with the specified name.

```
public double getDoubleProperty(java.lang.String name)
   throws JMSException
```

Name	Description
name	The name of the double property.

Throws JMSException if JMS fails to get Property due to some internal JMS error.

Throws MessageFormatException if this type conversion is invalid.

The getStringProperty Method

Returns the value of the String property with the specified name.

```
public java.lang.String getStringProperty(java.lang.String name)
   throws JMSException
```

Name	Description
name	The name of the String property.

Throws JMSException if JMS fails to get Property due to some internal JMS error.

Throws MessageFormatException if this type conversion is invalid.

The getObjectProperty Method

Returns the value of the Java object property with the specified name.

This method can be used to return, in objectified format, an object that has been stored as a property in the message with the equivalent setObjectProperty method call, or its equivalent primitive settypeProperty method.

```
public java.lang.Object getObjectProperty(java.lang.String name)
   throws JMSException
```

Name	Description
name	The name of the Java object property.

Throws JMSException if JMS fails to get Property due to some internal JMS error.

The getPropertyNames Method

Returns an Enumeration of all the property names.

Note: *The JMS standard header fields are not considered properties and are not returned in this enumeration.*

```
public java.util.Enumeration getPropertyNames()
   throws JMSException
```

Throws JMSException if JMS fails to get Property names due to some internal JMS error.

The setBooleanProperty Method

Sets a boolean property value with the specified name into the message.

```
public void setBooleanProperty(java.lang.String name,boolean value)
   throws JMSException
```

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

Throws JMSException if JMS fails to set Property due to some internal JMS error.

Throws MessageNotWriteableException - if properties are read-only

The setByteProperty Method

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

```
public void setByteProperty(java.lang.String name, byte value)
   throws JMSException
```

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

Throws JMSException if JMS fails to set Property due to some internal JMS error.

Throws MessageNotWriteableException if properties are read-only

The setShortProperty Method

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

```
public void setShortProperty(java.lang.String name, short value)
   throws JMSException
```

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

Throws JMSException if JMS fails to set Property due to some internal JMS error.

Throws MessageNotWriteableException if properties are read-only

The setIntProperty Method

Sets an int property value with the specified name into the message.

```
public void setIntProperty(java.lang.String name, int value)throws
   JMSException
```

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

Throws JMSException if JMS fails to set Property due to some internal JMS error.

Throws MessageNotWriteableException if properties are read-only

The setLongProperty Method

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

```
public void setLongProperty(java.lang.String name, long value)
   throws JMSException
```

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

Throws JMSException if JMS fails to set Property due to some internal JMS error.

Throws MessageNotWriteableException if properties are read-only

The setFloatProperty Method

Sets a float property value with the specified name into the message.

```
public void setFloatProperty(java.lang.String name, float value)
   throws JMSException
```

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

Throws JMSException if JMS fails to set Property due to some internal JMS error.

Throws MessageNotWriteableException if properties are read-only

The setDoubleProperty Method

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

```
public void setDoubleProperty(java.lang.String name, double value)
   throws JMSException
```

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

Throws JMSException if JMS fails to set Property due to some internal JMS error.

Throws MessageNotWriteableException if properties are read-only

The setStringProperty Method

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

```
public void setStringProperty(java.lang.String name, java.lang.String
value)
   throws JMSException
```

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

Throws JMSException if JMS fails to set Property due to some internal JMS error.

Throws MessageNotWriteableException if properties are read-only

The setObjectProperty Method

Sets a Java object property value with the specified name into the message.

Note: *This method only works for the objectified primitive object types (Integer, Double, Long ...) and Strings.*

```
public void setObjectProperty(java.lang.String name, java.lang.Object
value)
   throws JMSException
```

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

Throws JMSException if JMS fails to set Property due to some internal JMS error.

Throws MessageFormatException if object is invalid

Throws MessageNotWriteableException if properties are read-only

The acknowledge Method

Acknowledges all consumed messages of the session of this consumed message. All JMS messages support the acknowledge() method for use when a client has specified that a JMS consumers messages are to be explicitly acknowledged.

JMS defaults to implicit message acknowledgement. In this mode, calls to acknowledge() are ignored.

Acknowledgment of a message automatically acknowledges all messages previously received by the session. Clients may individually acknowledge messages or they may choose to acknowledge messages in application defined groups (which is done by acknowledging the last received message in the group).

Messages that have been received but not acknowledged may be redelivered to the consumer.

```
public void acknowledge()
   throws JMSException
```

Throws JMSException if JMS fails to acknowledge due to some internal JMS error.

Throws IllegalStateException if this method is called on a closed session.

The clearBody Method

Clears out the message body. Clearing a message's body does not clear its header values or property entries.

If this message body was read-only, calling this method leaves the message body in the same state as an empty body in a newly created message.

```
public void clearBody()
   throws JMSException
```

Throws JMSException if JMS fails to due to some internal JMS error.

7.3.20 interface javax.jms.BytesMessage

```
public interface BytesMessage
extends Message
```

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

The readBoolean Method

Reads a boolean from the bytes message stream.

```
public boolean readBoolean()
   throws JMSException
```

Throws MessageNotReadableException if message in write-only mode.

Throws JMSException if JMS fails to read message due to some internal JMS error.

Throws MessageEOFException if end of bytes stream

The readByte Method

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

```
public byte readByte()
   throws JMSException
```

Throws MessageNotReadableException if message in write-only mode.

Throws MessageEOFException if end of message stream

Throws JMSException if JMS fails to read message due to some internal JMS error.

The readUnsignedByte Method

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

```
public int readUnsignedByte()
   throws JMSException
```

Throws MessageNotReadableException if message in write-only mode.

Throws MessageEOFException if end of message stream

Throws JMSException if JMS fails to read message due to some internal JMS error.

The readShort Method

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

```
public short readShort()
   throws JMSException
```

Throws MessageNotReadableException if message in write-only mode.

Throws MessageEOFException if end of message stream

Throws JMSException if JMS fails to read message due to some internal JMS error.

The readUnsignedShort Method

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

```
public int readUnsignedShort()
   throws JMSException
```

Throws MessageNotReadableException if message in write-only mode.

Throws MessageEOFException if end of message stream

Throws JMSException if JMS fails to read message due to some internal JMS error.

The readChar Method

Reads a Unicode character value from the bytes message stream.

```
public char readChar()
   throws JMSException
```

Throws MessageNotReadableException if message in write-only mode.

Throws MessageEOFException if end of message stream .

Throws JMSException if JMS fails to read message due to some internal JMS error.

The readInt Method

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

```
public int readInt()
   throws JMSException
```

Throws MessageNotReadableException if message in write-only mode.

Throws MessageEOFException if end of message stream

Throws JMSException if JMS fails to read message due to some internal JMS error.

The readLong Method

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

```
public long readLong()
   throws JMSException
```

Throws MessageNotReadableException if message in write-only mode.

Throws MessageEOFException if end of message stream

Throws JMSException if JMS fails to read message due to some internal JMS error.

The readFloat Method

Reads a float from the bytes message stream.

```
public float readFloat()
   throws JMSException
```

Throws MessageNotReadableException if message in write-only mode.

Throws MessageEOFException if end of message stream

Throws JMSException if JMS fails to read message due to some internal JMS error.

The readDouble Method

Reads a double from the bytes message stream.

```
public double readDouble()
   throws JMSException
```

Throws MessageNotReadableException if message in write-only mode.

Throws MessageEOFException if end of message stream .

Throws JMSException if JMS fails to read message due to some internal JMS error.

The readUTF Method

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

```
public java.lang.String readUTF()
   throws JMSException
```

Throws MessageNotReadableException if message in write-only mode.

Throws MessageEOFException if end of message stream .

Throws JMSException if JMS fails to read message due to some internal JMS error.

The readBytes Method

Reads a byte array from the bytes message stream. If the length of array value is less than the bytes remaining to be read from the stream, the array should be filled. A subsequent call reads the next increment, and so on.

If the bytes remaining in the stream is less than the length of array value, the bytes should be read into the array. The return value of the total number of bytes read will be less than the length of the array, indicating that there are no more bytes left to be read from the stream. The next read of the stream returns -1.

```
public int readBytes(byte[] value)
   throws JMSException
```

Name	Description
value	The buffer into which the data is read.

Throws MessageNotReadableException if message in write-only mode.

Throws JMSException if JMS fails to read message due to some internal JMS error.

The readBytes Method

Reads a portion of the bytes message stream. If the length of array value is less than the bytes remaining to be read from the stream, the array should be filled. A subsequent call reads the next increment, etc.

If the bytes remaining in the stream is less than the length of array value, the bytes should be read into the array. The return value of the total number of bytes read will be less than the length of the array, indicating that there are no more bytes left to be read from the stream. The next read of the stream returns -1.

If length is negative, or length is greater than the length of the array value, then an IndexOutOfBoundsException is thrown. No bytes will be read from the stream for this exception case.

```
public int readBytes(byte[] value, int length)
   throws JMSException
```

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

Throws MessageNotReadableException if message in write-only mode.

Throws JMSException if JMS fails to read message due to some internal JMS error.

The writeBoolean Method

Writes a boolean to the bytes message stream as a 1-byte value. The value true is written as the value (byte)1; the value false is written as the value (byte)0.

```
public void writeBoolean(boolean value)
   throws JMSException
```

Name	Description
value	The `boolean` value to be written.

Throws MessageNotWriteableException if message in read-only mode.

Throws JMSException if JMS fails to write message due to some internal JMS error.

The writeByte Method

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

```
public void writeByte(byte value)
   throws JMSException
```

Name	Description
value	The byte value to be written.

Throws MessageNotWriteableException if message in read-only mode.

Throws JMSException if JMS fails to write message due to some internal JMS error.

The writeShort Method

Writes a short to the bytes message stream as two bytes, high byte first.

```
public void writeShort(short value)
   throws JMSException
```

Name	Description
value	The short to be written.

Throws MessageNotWriteableException if message in read-only mode.

Throws JMSException if JMS fails to write message due to some internal JMS error.

The writeChar Method

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

```
public void writeChar(char value)
   throws JMSException
```

Name	Description
value	The char value to be written.

Throws MessageNotWriteableException if message in read-only mode.

Throws JMSException if JMS fails to write message due to some internal JMS error.

The writeInt Method

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

```
public void writeInt(int value)
   throws JMSException
```

Name	Description
value	The int to be written.

Throws MessageNotWriteableException if message in read-only mode.

Throws JMSException if JMS fails to write message due to some internal JMS error.

The writeLong Method

Writes a long to the bytes message stream as eight bytes, high byte first.

```
public void writeLong(long value)
   throws JMSException
```

Name	Description
value	The long to be written.

Throws MessageNotWriteableException if message in read-only mode.

Throws JMSException if JMS fails to write message due to some internal JMS error.

The writeFloat Method

Converts the float argument to an int using the floatToIntBits method in class Float, and then writes that int value to the bytes message stream as a 4-byte quantity, high byte first.

```
public void writeFloat(float value)
   throws JMSException
```

Name	Description
value	The float value to be written.

Throws MessageNotWriteableException if message in read-only mode.

Throws JMSException if JMS fails to write message due to some internal JMS error.

The writeDouble Method

Converts the double argument to a long using the doubleToLongBits method in class Double, and then writes that long value to the bytes message stream as an 8-byte quantity, high byte first.

```
public void writeDouble(double value)
   throws JMSException
```

Name	Description
value	The double value to be written.

Throws MessageNotWriteableException if message in read-only mode.

Throws JMSException if JMS fails to write message due to some internal JMS error.

The writeUTF Method

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

```
public void writeUTF(java.lang.String value)
   throws JMSException
```

Name	Description
value	The String value to be written.

Throws MessageNotWriteableException if message in read-only mode.

Throws JMSException if JMS fails to write message due to some internal JMS error.

The writeBytes Method

Writes a byte array to the bytes message stream.

```
public void writeBytes(byte[] value)
   throws JMSException
```

Name	Description
value	The byte array to be written.

Throws MessageNotWriteableException if message in read-only mode.

Throws JMSException if JMS fails to write message due to some internal JMS error.

The writeBytes Method

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

```
public void writeBytes(byte[] value, int offset, int length)
   throws JMSException
```

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

Throws MessageNotWriteableException if message in read-only mode.

Throws JMSException if JMS fails to write message due to some internal JMS error.

The writeObject Method

Writes an object to the bytes message stream.

Note: *This method only works for the objectified primitive object types (Integer, Double, Long ...), Strings and byte arrays.*

```
public void writeObject(java.lang.Object value)
   throws JMSException
```

Name	Description
value	The object in the Java programming language ("Java object") to be written; it must not be null.

Throws NullPointerException if parameter value is null.

Throws MessageNotWriteableException if message in read-only mode.

Throws MessageFormatException if object is invalid type.

Throws JMSException if JMS fails to write message due to some internal JMS error.

The reset Method

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

```
public void reset()
   throws JMSException
```

Throws JMSException if JMS fails to reset the message due to some internal JMS error.

Throws MessageFormatException if message has an invalid format

7.3.21 interface javax.jms.MapMessage

```
public interface MapMessage
extends Message.
```

A MapMessage is used to send a set of name-value pairs, where names are Strings, and values are Java primitive types. The entries are accessed sequentially or randomly by name. The order of the entries is undefined. It inherits from Message, and adds a map message body.

The primitive types can be read or written explicitly using methods for each type. They may also be read or written generically as objects. For instance, a call to MapMessage.setInt("foo", 6) is equivalent to MapMessage.setObject("foo", new Integer(6)). Both forms are provided because the explicit form is convenient for static programming and the object form is needed when types are not known at compile time.

When a client receives a MapMessage, it is in read-only mode. At this time, if the client attempts to write to the message, a MessageNotWriteableException is thrown. If clearBody is called, the message can now be both read from and written to.

The getBoolean Method

Returns the boolean value with the specified name.

```
public boolean getBoolean(java.lang.String name)
   throws JMSException
```

Name	Description
name	The name of the boolean.

Throws JMSException if JMS fails to read message due to some internal JMS error.

Throws MessageFormatException if this type conversion is invalid.

The getByte Method

Returns the byte value with the specified name.

```
public byte getByte(java.lang.String name)
   throws JMSException
```

Name	Description
name	The name of the byte.

Throws JMSException if JMS fails to read message due to some internal JMS error.

Throws MessageFormatException if this type conversion is invalid.

The getShort Method

Returns the short value with the specified name.

```
public short getShort(java.lang.String name)
   throws JMSException
```

Name	Description
name	The name of the short.

Throws JMSException if JMS fails to read message due to some internal JMS error.

Throws MessageFormatException if this type conversion is invalid.

The getChar Method

Returns the Unicode character value with the specified name.

```
public char getChar(java.lang.String name)
   throws JMSException
```

Name	Description
name	The name of the Unicode character.

Throws JMSException if JMS fails to read message due to some internal JMS error.

Throws MessageFormatException if this type conversion is invalid.

The getInt Method

Returns the int value with the specified name.

```
public int getInt(java.lang.String name)
   throws JMSException
```

Name	Description
name	The name of the int.

Throws JMSException if JMS fails to read message due to some internal JMS error.

Throws MessageFormatException if this type conversion is invalid.

The getLong Method

Returns the long value with the specified name.

```
public long getLong(java.lang.String name)
   throws JMSException
```

Name	Description
name	The name of the long.

Throws JMSException if JMS fails to read message due to some internal JMS error.

Throws MessageFormatException if this type conversion is invalid.

The getFloat Method

Returns the float value with the specified name.

```
public float getFloat(java.lang.String name)
   throws JMSException
```

Name	Description
name	The name of the float.

Throws JMSException if JMS fails to read message due to some internal JMS error.

Throws MessageFormatException if this type conversion is invalid.

The getDouble Method

Returns the double value with the specified name.

```
public double getDouble(java.lang.String name)
   throws JMSException
```

Name	Description
name	The double value with the specified name..

Throws JMSException if JMS fails to read message due to some internal JMS error.

Throws MessageFormatException if this type conversion is invalid.

The getString Method

Returns the String value with the specified name.

```
public java.lang.String getString(java.lang.String name)
   throws JMSException
```

Name	Description
name	The name of the String.

Throws JMSException if JMS fails to read message due to some internal JMS error.

Throws MessageFormatException if this type conversion is invalid.

The getBytes Method

Returns the byte array value with the specified name.

```
public byte[] getBytes(java.lang.String name)
   throws JMSException
```

Name	Description
name	The name of the byte array.

Throws JMSException if JMS fails to read message due to some internal JMS error.

Throws MessageFormatException if this type conversion is invalid.

The getObject Method

Returns the Java object value with the specified name.

```
public java.lang.Object getObject(java.lang.String name)
   throws JMSException
```

Name	Description
name	The name of the Java object.

Throws JMSException if JMS fails to read message due to some internal JMS error.

The getMapNames Method

Returns an Enumeration of all the names in the MapMessage object.

```
public java.util.Enumeration getMapNames()
   throws JMSException
```

Throws JMSException if JMS fails to read message due to some internal JMS error.

The setBoolean Method

Sets a boolean value with the specified name into the Map.

```
public void setBoolean(java.lang.String name, boolean value)
   throws JMSException
```

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

Throws JMSException if JMS fails to read message due to some internal JMS error.

Throws MessageFormatException if this type conversion is invalid.

The setByte Method

Sets a byte value with the specified name into the Map.

```
public void setByte(java.lang.String name, byte value)
   throws JMSException
```

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

Throws JMSException if JMS fails to read message due to some internal JMS error.

Throws MessageNotWriteableException if the message is in read-only mode.

The setShort Method

Sets a short value with the specified name into the Map.

```
public void setShort(java.lang.String name, short value)
   throws JMSException
```

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

Throws JMSException if JMS fails to read message due to some internal JMS error.

Throws MessageNotWriteableException if the message is in read-only mode.

The setChar Method

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

```
public void setChar(java.lang.String name, char value)
   throws JMSException
```

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

Throws JMSException if JMS fails to read message due to some internal JMS error.

Throws MessageNotWriteableException if the message is in read-only mode.

The setInt Method

Sets an int value with the specified name into the Map.

```
public void setInt(java.lang.String name, int value)
   throws JMSException
```

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

Throws JMSException if JMS fails to read message due to some internal JMS error.

Throws MessageNotWriteableException if the message is in read-only mode.

The setLong Method

Sets a long value with the specified name into the Map.

```
public void setLong(java.lang.String name, long value)
   throws JMSException
```

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

Throws JMSException if JMS fails to read message due to some internal JMS error.

Throws MessageNotWriteableException if the message is in read-only mode.

The setFloat Method

Sets a float value with the specified name into the Map.

```
public void setFloat(java.lang.String name, float value)
   throws JMSException
```

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

Throws JMSException if JMS fails to read message due to some internal JMS error.

Throws MessageNotWriteableException if the message is in read-only mode.

The setDouble Method

Sets a double value with the specified name into the Map.

```
public void setDouble(java.lang.String name, double value)
   throws JMSException
```

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

Throws JMSException if JMS fails to read message due to some internal JMS error.

Throws MessageNotWriteableException if the message is in read-only mode.

The setString Method

Sets a String value with the specified name into the Map.

```
public void setString(java.lang.String name, java.lang.String value)
   throws JMSException
```

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

Throws JMSException if JMS fails to read message due to some internal JMS error.

Throws MessageNotWriteableException if the message is in read-only mode.

The setBytes Method

Sets a byte array value with the specified name into the Map.

```
public void setBytes(java.lang.String name, byte[] value)
   throws JMSException
```

Name	Description
name	The name of the byte array.
value	The byte array value to set in the Map. The array is copied so that the value for name will not be altered by future modifications.

Throws JMSException if JMS fails to read message due to some internal JMS error.

Throws MessageNotWriteableException if the message is in read-only mode.

The setBytes Method

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

```
public void setBytes(java.lang.String name, byte[] value, int offset,
int length)
   throws JMSException
```

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

Throws JMSException if JMS fails to read message due to some internal JMS error.

Throws MessageNotWriteableException if the message is in read-only mode.

The setObject Method

Sets a Java object value with the specified name into the Map.

```
public void setObject(java.lang.String name, java.lang.Object value)
   throws JMSException
```

Name	Description
name	The name of the Java object.
value	The Java object value to set in the Map.

Note: *This method only works for the obejectified primitive object types (Integer, Double, Long...) Strings and byte arrarys.*

Throws JMSException if JMS fails to read message due to some internal JMS error.

Throws MessageFormatException if the object is invalid.

Throws MessageNotWriteableException if the message is in read-only mode.

The itemExists Method

Queries whether an item exists in this MapMessage object.

```
public boolean itemExists(java.lang.String name)
   throws JMSException
```

Name	Description
name	The name of the item to test.

Throws JMSException if JMS fails to read message due to some internal JMS error.

7.3.22 interface javax.jms.ObjectMessage

public interface ObjectMessage

extends Message

An ObjectMessage is used to send a message that contains a serializable Java object. It inherits from Message and adds a body containing a single Java reference. Only Serializable Java objects can be used.

When a client receives an ObjectMessage, the object isin read-only mode. If an attemp is made to write to the message, a MessageNotWriteableException is thrown. If clearBody is called, the message can be both read from and written to.

The setObject Method

Sets the serializable object containing this message's data.

Important: *An ObjectMessage contains a snapshot of the object at the time setObject is called. Subsequent modifications of the object have no affect on the ObjectMessage body.*

```
public void setObject(java.io.Serializable object)
   throws JMSException
```

Name	Description
object	The message data.

Throws JMSException if JMS fails to read message due to some internal JMS error.

Throws MesssageFormatException if the object serialization fails.

Throws MessageNotWriteableException if the message is in read-only mode.

The getObject Method

Gets the serializable object containing this message's data. The default value is null.

```
public java.io.Serializable getObject()
   throws JMSException
```

Throws JMSException if JMS fails to read message due to some internal JMS error.

Throws MesssageFormatException if the object serialization fails.

7.3.23 interface javax.jms.StreamMessage

public interface StreamMessage

extends Message

A StreamMessage is used to send a stream of Java primitives. It is populated and read sequentially. It inherits from Message and adds a stream message body.

The primitive types can be read or written explicitly using methods for each type. They may also be read or written generically as objects. For instance, a call to StreamMessage.writeInt(6) is equivalent to StreamMessage.writeObject(new Integer(6)). Both forms are provided because the explicit form is convenient for static programming and the object form is needed when types are not known at compile time.

When the message is created, and also when clearBody is called, the body of the message is in write-only mode. After the first call to reset has been made, the message body is in read-only mode. When a message has been sent, by definition, the provider calls reset in order to read the content. When a message has been received, the provider calls reset, and sets the message body is in read-only mode for the client.

If clearBody is called on a message in read-only mode, the message body is cleared and the message body is in write-only mode. If a client attempts to read a message in write-only mode, a MessageNotReadableException is thrown. If a client attemps to write a message in read-only mode, a MessageNotWriteableException is thrown.

The readBoolean Method

Reads a boolean from the stream message.

```
public boolean readBoolean()
   throws JMSException
```

Throws JMSException if JMS fails to read message due to some internal JMS error.

Throws MessageEOFException if an end of message stream.

Throws MesssageFormatException if the object serialization fails.

Throws MessageNotWriteableException if the message is in read-only mode.

The readByte Method

Reads a byte value from the stream message.

```
public byte readByte()
   throws JMSException
```

Throws JMSException if JMS fails to read message due to some internal JMS error.

Throws MessageEOFException if an end of message stream.

Throws MesssageFormatException if the object serialization fails.

Throws MessageNotWriteableException if the message is in read-only mode.

The readShort Method

Reads a 16-bit integer from the stream message.

```
public short readShort()
   throws JMSException
```

Throws JMSException if JMS fails to read message due to some internal JMS error.

Throws MessageEOFException if an end of message stream.

Throws MesssageFormatException if the object serialization fails.

Throws MessageNotWriteableException if the message is in read-only mode.

The readChar Method

Reads a Unicode character value from the stream message.

```
public char readChar()
   throws JMSException
```

Throws JMSException if JMS fails to read message due to some internal JMS error.

Throws MessageEOFException if an end of message stream.

Throws MesssageFormatException if the object serialization fails.

Throws MessageNotWriteableException if the message is in read-only mode.

The readInt Method

Reads a 32-bit integer from the stream message.

```
public int readInt()
   throws JMSException
```

Throws JMSException if JMS fails to read message due to some internal JMS error.

Throws MessageEOFException if an end of message stream.

Throws MesssageFormatException if the object serialization fails.

Throws MessageNotWriteableException if the message is in read-only mode.

The readLong Method

Reads a 64-bit integer from the stream message.

```
public long readLong()
   throws JMSException
```

Throws JMSException if JMS fails to read message due to some internal JMS error.

Throws MessageEOFException if an end of message stream.

Throws MesssageFormatException if the object serialization fails.

Throws MessageNotWriteableException if the message is in read-only mode.

The readFloat Method

Reads a float from the stream message.

```
public float readFloat()
   throws JMSException
```

Throws JMSException if JMS fails to read message due to some internal JMS error.

Throws MessageEOFException if an end of message stream.

Throws MesssageFormatException if the object serialization fails.

Throws MessageNotWriteableException if the message is in read-only mode.

The readDouble Method

Reads a double from the stream message.

```
public double readDouble()
   throws JMSException
```

Throws JMSException if JMS fails to read message due to some internal JMS error.

Throws MessageEOFException if an end of message stream.

Throws MesssageFormatException if the object serialization fails.

Throws MessageNotWriteableException if the message is in read-only mode.

The readString Method

Reads a String from the stream message.

```
public java.lang.String readString()
   throws JMSException
```

Throws JMSException if JMS fails to read message due to some internal JMS error.

Throws MessageEOFException if an end of message stream.

Throws MesssageFormatException if the object serialization fails.

Throws MessageNotWriteableException if the message is in read-only mode.

The readBytes Method

Reads a byte array field from the stream message into the specified byte[] object (the read buffer).

To read the field value, readBytes should be successively called until it returns a value less than the length of the read buffer. The value of the bytes in the buffer following the last byte read is undefined.

If readBytes returns a value equal to the length of the buffer, a subsequent readBytes call must be made. If there are no more bytes to be read, this call returns -1.

If the byte array field value is null, readBytes returns -1.

If the byte array field value is empty, readBytes returns 0.

Once the first readBytes call on a byte[] field value has been made, the full value of the field must be read before it is valid to read the next field. An attempt to read the next field before that has been done will throw a MessageFormatException.

To read the byte field value into a new byte[] object, use the readObject method.

```
public int readBytes(byte[] value)
   throws JMSException
```

Name	Description
value	The buffer into which the data is read.

Throws JMSException if JMS fails to read message due to some internal JMS error.

Throws MessageEOFException if an end of message stream.

Throws MesssageFormatException if the object serialization fails.

Throws MessageNotWriteableException if the message is in read-only mode.

The readObject Method

Reads an object from the stream message. This method can be used to return in objectified format, an object that had been written to the Stream with the equivalent writeObject method call, or the equivalent primitive write method.

```
public java.lang.Object readObject()
   throws JMSException
```

Note: *Byte vlaues are returned as byte[], not Byte[].*

Throws JMSException if JMS fails to read message due to some internal JMS error.

Throws MessageEOFException if an end of message stream.

Throws MessageNotWriteableException if the message is in read-only mode.

The writeBoolean Method

Writes a boolean to the stream message. The value true is written as the value (byte)1; the value false is written as the value (byte)0.

```
public void writeBoolean(boolean value)
   throws JMSException
```

Throws JMSException if JMS fails to read message due to some internal JMS error.

Throws MessageNotWriteableException if the message is in read-only mode.

The writeByte Method

Writes a byte to the stream message.

```
public void writeByte(byte value)
   throws JMSException
```

Name	Description
value	The byte value to be written.

Throws JMSException if JMS fails to read message due to some internal JMS error.

Throws MessageNotWriteableException if the message is in read-only mode.

The writeShort Method

Writes a short to the stream message.

```
public void writeShort(short value)
   throws JMSException
```

Name	Description
value	The short value to be written.

Throws JMSException if JMS fails to read message due to some internal JMS error.

Throws MessageNotWriteableException if the message is in read-only mode.

The writeChar Method

Writes a char to the stream message.

```
public void writeChar(char value)
   throws JMSException
```

Name	Description
value	The char value to be written.

Throws JMSException if JMS fails to read message due to some internal JMS error.

Throws MessageNotWriteableException if the message is in read-only mode.

The writeInt Method

Writes an int to the stream message.

```
public void writeInt(int value)
   throws JMSException
```

Name	Description
value	The int value to be written.

Throws JMSException if JMS fails to read message due to some internal JMS error.

Throws MessageNotWriteableException if the message is in read-only mode.

The writeLong Method

Writes a long to the stream message.

```
public void writeLong(long value)
   throws JMSException
```

Name	Description
value	The long value to be written.

Throws JMSException if JMS fails to read message due to some internal JMS error.

Throws MessageNotWriteableException if the message is in read-only mode.

The writeFloat Method

Writes a float to the stream message.

```
public void writeFloat(float value)
   throws JMSException
```

Name	Description
value	The float value to be written.

Throws JMSException if JMS fails to read message due to some internal JMS error.

Throws MessageNotWriteableException if the message is in read-only mode.

The writeDouble Method

Writes a double to the stream message.

```
public void writeDouble(double value)
   throws JMSException
```

Name	Description
value	The double value to be written.

Throws JMSException if JMS fails to read message due to some internal JMS error.

Throws MessageNotWriteableException if the message is in read-only mode.

The writeString Method

Writes a string to the stream message.

```
public void writeString(java.lang.String value)
   throws JMSException
```

Name	Description
value	The String value to be written.

Throws JMSException if JMS fails to read message due to some internal JMS error.

Throws MessageNotWriteableException if the message is in read-only mode.

The writeBytes Method

Writes a byte array field to the stream message. The byte array value is written as a byte array field into the StreamMessage. Consecutively written byte array fields are treated as two distinct fields when reading byte array fields.

```
public void writeByte(byte[] value)
   throws JMSException
```

Name	Description
value	The byte array to be written.

Throws JMSException if JMS fails to read message due to some internal JMS error.

Throws MessageNotWriteableException if the message is in read-only mode.

The writeBytes Method

Writes a portion of a byte array as a byte array field to the stream message. The portionof the byte array value is written as a byte array field into the StreamMessage. Consecutively written byte array fields are treated as two distinct fields when reading byte array fields.

```
public void writeBytes(byte[] value, int offset, int length)
   throws JMSException
```

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

Throws JMSException if JMS fails to read message due to some internal JMS error.

Throws MessageNotWriteableException if the message is in read-only mode.

The writeObject Method

Writes a Java object to the stream message. This method only works for the objectified primitive object types (Integer, Double, Long..), Strings and byte arrays.

```
public void writeObject(java.lang.Object value)
   throws JMSException
```

Name	Description
value	The Java object to be written.

Throws JMSException if JMS fails to read message due to some internal JMS error.

Throws MessageNotWriteableException if the message is in read-only mode.

Throws MessageFormatException if the object is invalid.

The reset Method

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

```
public void reset()
   throws JMSException
```

Throws JMSException if JMS fails to read message due to some internal JMS error.

Throws MessageFormatException if the message has an invalid format.

7.3.24 interface javax.jms.TextMessage

public interface TextMessage

extends Message

A TextMessage is used to send a message containing a java.lang.String. It inherits from Message and adds a text message body.

When a client receives a TextMessage, it is in read-only mode. If an attempt is made to write to the message, while in read-only mode, a MessageNotWriteableException is thrown. If clearBody is called, the messae can be then read from and written to.

Refer to **interface javax.jms.Message** on page 135

The getText Method

Gets the string containing the data associated with the message. The default value is null.

```
public java.lang.String getText()
   throws JMSException
```

Throws JMSException if JMS fails to read message due to some internal JMS error.

The setText Method

Sets the string containing the data associated with the message.

```
public void setText(java.lang.String string)
   throws JMSException
```

Name	Description
string	The String containing the message data.

Throws JMSException if JMS fails to read message due to some internal JMS error.

Throws MessageNotWriteableException if the message is in read-only mode.

7.3.25 interface javax.jms.MessageConsumer

public interface MessageConsumer

A client uses a MessageConsumer object to receive messages from a destination. A MessageConsumer object is created by passing a Destination object to a message-consumer creation method supplied by a session.

MessageConsumer is the parent interface for all message consumers.

A message consumer can be created with a message selector. This allows the client to restrict the messges delivered to the message consumer to those that match the selector criteria.

A client may either synchronously receive a message consumer's messages, or have the consumer asynchronously deliver them as they arrive. A client can request the next message from a message consumer using one of the associated receive methods. There are several variations of receive that allow a client to poll, or wait for the next message.

A client can register a messageListener object with a message consumer. As messages arrive at the message consumer, it delivers them by calling the MessageListener's onMessage method.

It is a client programming error for a MessageListener to throw and exception.

The getMessageSelector Method

Gets this message consumer's message selector expression.

```
public java.lang.String getMessageSelector()
   throws JMSException
```

Throws JMSException if JMS fails to get the message selector due to some JMS error.

The getMessageListener Method

Gets the message consumer's `MessageListener`.

```
public MessageListener getMessageListener()
   throws JMSException
```

Throws JMSException if JMS fails to get the message listener due to some JMS error.

The setMessageListener Method

Sets the message consumer's MessageListener. Setting the message listener to null is the equivalent of unsetting the message listener for the message consumer.

Calling the setMessageListener method of MessageConsumer while messages are being consumed by an existing listener, or the consumer is being used to synchronously consume messages is undefined.

```
public void setMessageListener(MessageListener listener)
   throws JMSException
```

Name	Description
listener	The listener that the messages are to be delivered to.

Throws JMSException if the JMS fails to set the message listener due to some JMS error.

The receive Method

Receives the next message produced for this message consumer. This call blocks indefinitely until a message is produced or until this message consumer is close. If this receive os dpme wotjom a transaction, the message remains on the consumer until the trasaction commits.

```
public Message receive()
   throws JMSException
```

Throws JMSException if JMS fails to receive the next message due to some error.

The receive Method

Receives the next message that arrives within the specified timeout interval. This call blocks until a message arrives, the timeout expires, or this message consumer is closed. A timeout of zero never expires and the call blocks indefinitely.

```
public Message receive(long timeout)
   throws JMSException
```

Name	Description
timeout	The timeout value (in milliseconds).

The receiveNoWait Method

Receives the next message if one is immediately available.

```
public Message receiveNoWait()
   throws JMSException
```

Throws JMSExceptio if JMS fails to receive the next message due to some error.

The close Method

Closes the message consumer. Since a provider may allocate some resources on behalf of a MessageConsumer outside the JVM, clients should close them when they are not needed. Relying on garbage collection to eventually reclaim these resources may not be effective enough.

This call blocks until a receive or message listener in progress has completed. A blocked message consumer receive call returns null when this message consumer is close.

```
public void close()
   throws JMSException
```

Throws JMSException if JMS fails to close the consumer due to some error.

7.3.26 interface javax.jms.QueueReceiver

public interface QueueReceiver

extends MessageConsumer

A client uses a QueueReceiver for receiving messages that have been delivered to a queue. Although it is possible to have multiple QueueReceivers for the same queue, JMS does not define how messages are distributed between the QueueReceivers.

The getQueue Method

Get the queue associated with this queue receiver.

```
public Queue getQueue()
               throws JMSException
```

Throws JMSException if JMS fails to get queue for this queue receiver due to some internal error.

7.3.27 interface javax.jms.TopicSubscriber

public interface TopicSubscriber

extends MessageConsumer.

A client uses a TopicSubscriber for receiving messages that have been published to a topic. TopicSubscriber is the Pub/Sub variant of a JMS message consumer.

A topic session allows for the creation of multiple topic subscribers per Destination. It delivers each message for a destionation to each topic subscriber that is eligible to receive it. Each copy of the message is treated as a completely separate message. Work performed on one copy has no affect on another, acknowledging one does not acknowledge the other, one message may be delivered immediately, while another waits for the consumer to process messages ahead of it.

Regular TopicSubscribers are not durable. They only receive messages that are published while they are active. Messages filtered out by a subscriber's message selector, will never be delivered to the subscriber. From the subscriber's perspective, they do not exist.

In some cases, a connection both publishes and subscribes to a topic. The subscriber NoLocal attribute allows a subscriber to inhibit the delivery of messages published by its own connection.

If a client needs to receive all of the messages published on a topic, including those published while the subscriber is inactive, a durable TopicSubscriber is used. JMS retains a record of this durable subscription and insures that all messages from the topic's associated publishers are retained until they are either acknowledged by the durable subscriber, or they have expired.

Sessions with durable subscribers must always provide the same client identifier. In addition, each client must specify a name that uniquely identifies (within the client identifier) each durable subscription it creates. Only one session at a time can have a TopicSubscriber for particular durable subscription.

A client can change an exiting durable subscription, by creating a durable TopicSubscriber with the same name, and a new topic and/or message selector. Changing a durable subscription is equivalent to delting and recreating it.

TopicSessions provide the unsubscribe method for deleting a durable subscription created by their client. This deletes the state being maintained on behalf of the subscriber by its provider.

Refer to **interface javax.jms.MessageConsumer** on page 174

The getTopic Method

```
public Topic getTopic()
               throws JMSException
```

Get the topic associated with this subscriber.

Throws JMSException if JMS fails to get topic for this topic subscriber due to some internal error.

The getNoLocal Method

```
public boolean getNoLocal()
                   throws JMSException
```

Get the NoLocal attribute for this TopicSubscriber. The default value for this attribute is false.

Throws JMSException if JMS fails to get noLocal attribute for this topic subscriber due to some internal error.

7.3.28 interface javax.jms.MessageListener

```
public interface MessageListener
```

A MessageListener is used to receive asynchronously delivered messages. Each session must insure that it passes messages serially to the listener. This means that a listener assigned to one or more consumers of the same session, can assume that the onMessage method is not called with the next message until the session has completed the last call.

The onMessage Method

Passes a message to the listener.

```
public void onMessage(Message message)
```

Name	Description
message	The message passed to the listener.

7.3.29 interface javax.jms.MessageProducer

```
public interface MessageProducer
```

A client uses a message producer to send messages to a Destination. The message is created by passing a Destination to a create message producer method, supplied by a Session.

A client also can optionally create a message producer, without supplying a Destination. In this case, a Destination must be input on every send operation. A typical use for this style of message producer, is to send replies to requests, using the request's replyToDestination.

A client can specify a time-to-live value, in milliseconds, for each message sent. This value defines a message expiration time, which is the sum of the message's time-to-live, and the GMT at which it is sent (for trasacted sends, this is the time the client sends the message, not the time the transaction is committed).

A JMS provider should attempt to accurately expire message, as the means to acquire this accuracy is not pre-defined.

The setDisableMessageID Method

Sets whether message IDs are disabled. Since message IDs take some effort to create, and increase the size of a message, some JMS providers may choose to optimize message overhead, if they suspect that the message ID is not going to be used by an application. JMS message Producers provide a hint to disable message ID. When a client sets a Producer to disable message ID, they are saying that they do not depend on the value of the message ID for the messages it then produces. These messages must either have the message ID set to null, or if the hint is ignored, the message ID must be set to the normal unique value. Message IDs are enabled by default.

```
public void setDisableMessageID(boolean value)
   throws JMSException
```

Name	Description
value	Indicates if this messages identifiers are disabled.

Throws JMSException if JMS fails to set the disabled message ID due to some internal error.

The getDisableMessageID Method

Gets an indication of whether message IDs are disabled.

```
public boolean getDisableMessageID()
   throws JMSException
```

Throws JMSException if JMS fails to get the disabled message ID due to some internal error.

The setDisableMessageTimestamp Method

Sets whether message timestamps are disabled. Since timestamps require effort to create and increase a message's size, some JMS providers may optimize overhead by not enabling the timestamp, if they suspect that it is not going to be used by an application. JMS message Producers provide a hint to disable timestamps. When a client sets a producer to disable timestamps, they are not depending on the value of the timestamp, for the messages produced. These messages must either have timestamp set to null, or if the hint is ignored, the timestamp must be set to its normal value. Message timestamps are enabled by default.

```
public void setDisableMessageTimestamp(boolean value)
   throws JMSException
```

Name	Description
value	Indicates if this messages timestamps are disabled.

Throws JMSException if JMS fails to set the disabled message timestamp due to some internal error.

The getDisableMessageTimestamp Method

Gets an indication of whether message timestamps are disabled.

```
public boolean getDisableMessageTimestamp()
   throws JMSException
```

Throws JMSException if JMS fails to get and inciation of whether the message timestamp is disabled due to some internal error.

The setDeliveryMode Method

Sets the producer's default delivery mode. Delivery mode is set to PERSISTENT by default.

```
public void setDeliveryMode(int deliveryMode)
   throws JMSException
```

Name	Description
deliveryMode	The message delivery mode for this message producer; acceptable values are DeliveryMode.NON_PERSISTENT and DeliveryMode.PERSISTENT.

Throws JMSException if JMS fails to set delivery mode due to some internal error.

The getDeliveryMode Method

Gets the producer's default delivery mode.

```
public int getDeliveryMode()
   throws JMSException
```

Throws JMSException if JMS fails to get the delivery mode due to some internal error.

The setPriority Method

Sets the producer's default priority. The JMS API defines ten levels of priority value, with 0 as the lowest priority and 9 as the highest. Clients should consider priorities 0-4 as gradations of normal priority, and priorities 5-9 as gradations of expedited priority. Priority is set to 4 by default.

```
public void setPriority(int defaultPriority)
   throws JMSException
```

Name	Description
defaultPriority	The message priority for this message producer; must be a value between 0 and 9.

Throws JMSException if JMS fails to set the priority due to some internal error.

The getPriority Method

Gets the producer's default priority.

```
public int getPriority()
   throws JMSException
```

Throws JMSException if JMS fails to get the priority due to some internal error.

The setTimeToLive Method

Sets the default length of time, in milliseconds, from its dispatch, time that a produced message should be retained by the message system. Time-to-live is set to zero by default.

```
public void setTimeToLive(long timeToLive)
  throws JMSException
```

Name	Description
timeToLive	The message time to live in milliseconds; zero is unlimited.

Throws JMSException if JMS fails to set Time to Live due to some internal error.

The getTimeToLive Method

Gets the default length of time, in milliseconds, from its dispatch, time that a produced message should be retained by the message system.

```
public void getTimeToLive()
   throws JMSException
```

Throws JMSException if JMS fails to get Time to Live due to some internal error.

The close Method

```
public void close()
   throws JMSException
```

Since a provider may allocate some resources on behalf of a MessageProducer outside the JVM, clients should close them when they are not needed. Relying on garbage collection to eventually reclaim these resources may not be timely enough.

Throws JMSException if JMS fails to close the producer due to some error.

7.3.30 interface javax.jms.QueueSender

```
public interface QueueSender
extends MessageProducer
```

A client uses a QueueSender to send messages to a queue. Normally the Queue is specified when a QueueSender is created and in this case, attempting to use the methods for an unidentified QueueSender will throws an UnsupportedOperationException. In the case that the QueueSender with an unidentified Queue is created, the methods that assume the Queue has been identified throw an UnsupportedOperationException.

The getQueue Method

Get the queue associated with this queue sender.

```
public Queue getQueue()
               throws JMSException
```

Throws JMSException if JMS fails to get queue for this queue sender due to some internal error.

The send Method

Send a message to the queue. Use the QueueSender's default delivery mode, timeToLive and priority.

```
public void send(Message message)
          throws JMSException
```

Name	Description
message	The message to be sent

Throws JMSException if JMS fails to send the message due to some internal error.

Throws MessageFormatException if invalid message specified

Throws InvalidDestinationException if a client uses this method with a Queue sender with an invalid queue.

Thre send Method

Send a message specifying delivery mode, priority and time to live to the queue.

```
public void send(Message message, int deliveryMode, int priority,
long timeToLive)
   throws JMSException
```

Name	Description
message	The message to be sent
deliveryMode	The delivery mode to use.
priority	The priority for this message.
timeToLive	The message’s lifetime (in milliseconds)

Throws JMSException if JMS fails to send the message due to some internal error.

Throws MessageFormatException if invalid message specified

Throws InvalidDestinationException if a client uses this method with a Queue sender with an invalid queue.

The send Method

Send a message to a queue for an unidentified message producer. Use the QueueSender's default delivery mode, timeToLive and priority.

Typically a JMS message producer is assigned a queue at creation time; however, JMS also supports unidentified message producers which require that the queue be supplied on every message send.

```
public void send(Queue queue, Message message)
   throws JMSException
```

Name	Description
queue	The queue to which this message should be sent.
message	The message to be sent.

Throws JMSException if JMS fails to send the message due to some internal error.

Throws MessageFormatException if invalid message specified

Throws InvalidDestinationException if a client uses this method with an invalid queue.

The send Method

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

Typically a JMS message producer is assigned a queue at creation time; however, JMS also supports unidentified message producers which require that the queue be supplied on every message send.

```
public void send(Queue queue, Message message, int deliveryMode, int
priority, long timeToLive)
   throws JMSException
```

Name	Description
queue	The queue to which this message should be sent.
message	The message to be sent.
deliveryMode	The delivery mode to use.
priority	The priority for this message.
timeToLive	The message's lifetime (in milliseconds).

Throws JMSException if JMS fails to send the message due to some internal error.

Throws MessageFormatException if invalid message specified

Throws InvalidDestinationException if a client uses this method with an invalid queue.

7.3.31 interface javax.jms.TopicPublisher

```
public interface TopicPublisher
extends MessageProducer
```

A client uses a TopicPublisher for publishing messages on a topic. TopicPublisher is the Pub/Sub variant of a JMS message producer. Normally the Topic is specified when a TopicPublisher is created and in this case, attempting to use the methods for an unidentified TopicPublisher will throws an UnsupportedOperationException.

In the case that the TopicPublisher with an unidentified Topic is created, the methods that assume the Topic has been identified throw an UnsupportedOperationException.

The getTopic Method

Get the topic associated with this publisher.

```
public Topic getTopic()
   throws JMSException
```

Throws JMSException if JMS fails to get topic for this topic publisher due to some internal error.

The publish Method

Publish a Message to the topic Use the topics default delivery mode, timeToLive and priority.

```
public void publish(Message message)
   throws JMSException
```

Name	Description
message	The message to publish.

Throws JMSException if JMS fails to publish the message due to some internal error.

Throws MessageFormatException if invalid message specified.

Throws InvalidDestinationException if a client uses this method with a Topic Publisher with an invalid topic.

The publish Method

Publish a Message to the topic specifying delivery mode, priority and time to live to the topic.

```
public void publish(Message message, int deliveryMode, int priority,
long timeToLive)
   throws JMSException
```

Name	Description
message	The message to publish.
deliveryMode	The delivery mode to use.
priority	The priority for this message.
timeToLive	The message's lifetime (in milliseconds).

Throws JMSException if JMS fails to publish the message due to some internal error.

Throws MessageFormatException if invalid message specified

Throws InvalidDestinationException if a client uses this method with a Topic Publisher with an invalid topic.

The publish Method

Publish a Message to a topic for an unidentified message producer. Use the topics default delivery mode, timeToLive and priority.

Typically a JMS message producer is assigned a topic at creation time; however, JMS also supports unidentified message producers which require that the topic be supplied on every message publish.

```
public void publish(Topic topic, Message message)
   throws JMSException
```

Name	Description
topic	The topic to which to publish the message.
message	The message to publish.

Throws JMSException if JMS fails to publish the message due to some internal error.

Throws MessageFormatException if invalid message specified.

Throws InvalidDestinationException if a client uses this method with an invalid topic.

The publish Method

Publishes a Message to a topic for an unidentified message producer, specifying delivery mode, priority and time to live. Typically a JMS message producer is assigned a topic at creation time; however, JMS also supports unidentified message producers which require that the topic be supplied on every message publish.

```
public void publish(Topic topic, Message message,int deliveryMode,
int priority, long timeToLive)
   throws JMSException
```

Name	Description
topic	The topic to which to publish this message.
message	The message to be sent.
deliveryMode	The delivery mode to use.
priority	The priority for this message.
timeToLive	The message's lifetime (in milliseconds).

Throws JMSException if JMS fails to publish the message due to some internal error.

Throws MessageFormatException if invalid message specified

Throws InvalidDestinationException if a client uses this method with an invalid topic.

7.3.32 interface java.lang.Runnable

7.3.33 interface javax.jms.Session

```
public interface Session
   extends java.lang.Runnable
```

A Session object is a single-threaded context for producing and consuming messages. Although it may allocate provider resources outside the Java Virtual Machine (JVM), it is considered a lightweight JMS object.

A session serves several purposes:

- It is a factory for its message producers and consumers.
- It supplies provider-optimized message factories.
- It supports a single series of transactions that combine work spanning its producers and consumers into atomic units.
- It defines a serial order for the messages it consumes and the messages it produces.
- It retains messages it consumes until they have been acknowledged.
- It serializes execution of message listeners registered with its message consumers.

A session can create and service multiple message producers and consumers.

One typical use is to have a thread block on a synchronous MessageConsumer, until a message arrives. The thread may then use one or more of the Session's Message Producers.

For a client to have one thread producing messages, while others consume them, the client should use a separate Session for the producing thread.

Once a connection has been established, any session with a registered listener(s) is dedicated to the thread of control that delivers messages to it. It is erroneous for the client code to use this session, or any of it's constituent objects from another thread of control. The only exception to this, is the use of the session or connection close method.

Most clients can partition their work naturally into Sessions. This model allows clients to start simply, and incrementally, adding message processing complexity as the need for concurrency grows.

The close method is the only session method, that can be called while some other session method is being executed in another thread.

A session may be optionally specified as transacted. Each transacted session supports a single series of transactions. Each transaction groups a set of message sends, and a set of message receives, into an atomic unit of work. In effect, transactions organize a session's input message stream, and output message stream, into a series of atomic units. When a transaction commits, the atomic unit of input is acknowledged, and the associated atomic unit of output is sent. If a transaction rollback is performed, the associated sent messages are destroyed, and the session's input is automatically recovered.

The content of a transaction's input and output units, is that of the messages that have been produced and consumed within the session's current transaction. A transaction is completed by using either the session's commit or rollback method. The completion of a session's current transaction automatically begins the next. In this manner, a transacted session always has a current transaction within which the work is done.

The createBytesMessage Method

Creates a BytesMessage object. A BytesMessage object is used to send a message containing a stream of uninterpreted bytes.

```
public BytesMessage createBytesMessage()
   throws JMSException
```

Throws JMSException if JMS fails to create this message due to some internal error.

The createMapMessage Method

Creates a MapMessage object. A MapMessage object is used to send a self-defining set of name-value pairs, where names are String objects and values are primitive values in the Java programming language.

```
public MapMessage createMapMessage()
   throws JMSException
```

Throws JMSException if JMS fails to create this message due to some internal error.

The createMessage Method

Creates a Message object. The Message interface is the root interface of all JMS messages. A Message object holds all the standard message header information. It can be sent when a message containing only header information is sufficient.

```
public Message createMessage()
   throws JMSException
```

Throws JMSException if JMS fails to create this message due to some internal error.

The createObjectMessage Method

Creates an ObjectMessage object. An ObjectMessage object is used to send a message that contains a serializable Java object.

```
public ObjectMessage createObjectMessage()
   throws JMSException
```

Throws JMSException if JMS fails to create this message due to some internal error.

The createObjectMessage Method

Creates an initialized ObjectMessage object. An ObjectMessage object is used to send a message that contains a serializable Java object.

```
public ObjectMessage createObjectMessage(java.io.Serializable object)
   throws JMSException
```

Name	Description
object	The object to use to initialize this message.

Throws JMSException if JMS fails to create this message due to some internal error.

The createStreamMessage Method

Creates a StreamMessage object. A StreamMessage object is used to send a self-defining stream of primitive values in the Java programming language.

```
public StreamMessage createStreamMessage()
   throws JMSException
```

Throws JMSException if JMS fails to create this message due to some internal error.

The createTextMessage Method

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

```
public TextMessage createTextMessage()
   throws JMSException
```

Throws JMSException if JMS fails to create this message due to some internal error.

The createTextMessage Method

Creates an initialized TextMessage object. A TextMessage object is used to send a message containing a String.

```
public TextMessage createTextMessage(java.lang.String text)
   throws JMSException
```

Name	Description
text	The string used to initialize this message.

Throws JMSException if JMS fails to create this message due to some internal error.

The getTransacted Method

Queries whether the session is in transacted mode.

```
public boolean getTransacted()
   throws JMSException
```

Throws JMSException if JMS fails to return the transaction mode due to internal error in JMS Provider.

The commit Method

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

```
public void commit()
   throws JMSException
```

Throws JMSException if the JMS implementation fails to commit the the transaction due to some internal error.

Throws TransactionRolledBackException if the transaction gets rolled back due to some internal error during commit.

Throws IllegalStateException if the method is not called by a transacted session.

The rollback Method

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

```
public void rollback()
   throws JMSException
```

Throws JMSException if the JMS implementation fails to rollback the the transaction due to some internal error.

Throws IllegalStateException if the method is not called by a transacted session.

The close Method

Closes the session. A provider may allocate some resources on behalf of a Session outside the JVM, clients therefore, should close them when they are not needed. Relying on garbage collection to eventually reclaim these resources may not be timely enough.

There is no need to close the producers and consumers of a closed session. This call will block until a receive or message listener in progress has completed. A blocked message consumer receive call returns null when this session is closed. Closing a transacted session must rollback the in-progress transaction. This method is the only session method that can be concurrently called.

Invoking any other session method on a closed session must throw JMSException.IllegalStateException. Closing a closed session must NOT throw an exception.

```
public void close()
   throws JMSException
```

Throws JMSException if JMS implementation fails to close a Session due to some internal error.

The recover Method

Stops message delivery in this session, and restarts sending messages with the oldest unacknowledged message. All consumers deliver messages in a serial order. Acknowledging a received message automatically acknowledges all messages that have been delivered to the client.

Restarting a session causes it to take the following actions:

- Stops message delivery
- Marks all messages that might have been delivered but not acknowledged as `redelivered'
- Restart the delivery sequence including all unacknowledged messages that had been previously delivered.

Redelivered messages do not have to be delivered in exactly their original delivery order.

Throws JMSException if JMS implementation fails to stop message delivery and restart message send due to due to some internal error.

Throws IllegalStateException if method is called by a transacted session.

```
public void recover()
   throws JMSException
```

getMessageListener

Return the session's distinguished message listener (optional).

Throws JMSException if JMS fails to get the message listener due to an internal error in JMS Provider.

```
public MessageListener getMessageListener()
   throws JMSException
```

Throws JMSException if JMS fails to get the message listener due to an internal error in JMS Provider.

The setMessageListener Method

Sets the session's distinguished message listener (optional). When it is set, no other form of message receipt in the session can be used; however, all forms of sending messages are still supported. This is an expert facility not used by regular JMS clients.

```
public void setMessageListener(MessageListener listener)
   throws JMSException
```

Name	Description
listener	The message listener to associate with this session.

Throws JMSException if JMS fails to set the message listener due to an internal error in JMS Provider.

run

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

```
public void run()
```

Specified by

run in interface java.lang.Runnable

7.3.34 interface javax.jms.QueueSession

```
public interface QueueSession
extends Session
```

A QueueSession provides methods for creating QueueReceiver's, QueueSender's, QueueBrowser's and TemporaryQueues. If there are messages that have been received but not acknowledged when a QueueSession terminates, these messages will be retained and redelivered when a consumer next accesses the queue.

The createQueue Method

Creates a queue identity given a Queue name. This facility is provided for the rare cases where clients need to dynamically manipulate queue identity. This allows the creation of a queue identity with a provider specific name. Clients that depend on this ability are not portable.

Note: *This method is not for creating the physical topic. The physical creation of topics is an administration task and not to be initiated by the JMS interface. The one exception is the creation of temporary topics is done using the createTemporaryTopic method.*

```
public Queue createQueue(java.lang.String queueName)
   throws JMSException
```

Name	Description
queueName	The name of this queue.

Throws JMSException if a session fails to create a queue due to some JMS error.

The createReceiver Method

Creates a QueueReceiver to receive messages from the specified queue.

```
public QueueReceiver createReceiver(Queue queue)
   throws JMSException
```

Name	Description
queue	The name of the queue to access.

Throws JMSException - if a session fails to create a receiver due to some JMS error.

Throws InvalidDestinationException if invalid Queue specified.

The createReceiver Method

Creates a QueueReceiver to receive messages from the specified queue.

```
public QueueReceiver createReceiver(Queue queue, java.lang.String
messageSelector)
   throws JMSException
```

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

Throws JMSException if a session fails to create a receiver due to some JMS error.

Throws InvalidDestinationException if invalid Queue specified.

Throws InvalidSelectorException if the message selector is invalid.

The createSender Method

Creates a QueueSender to send messages to the specified queue.

```
public QueueSender createSender(Queue queue)
   throws JMSException
```

Name	Description
queue	The name of the queue to access, or null if this is anunidentified producer.

Throws JMSException if a session fails to create a sender due to some JMS error.

Throws InvalidDestinationException if invalid Queue specified.

The createTemporaryQueue Method

Creates a temporary queue. It's lifetime will be that of the QueueConnection unless deleted earlier.

```
public TemporaryQueue createTemporaryQueue()
   throws JMSException
```

Throws JMSException if a session fails to create a Temporary Queue due to some JMS error.

7.3.35 interface javax.jms.TopicSession

```
public interface TopicSession
extends Session
```

A TopicSession provides methods for creating TopicPublishers, TopicSubscribers and TemporaryTopics. Also provided are methods for deleting the associated client's durable subscribers.

The createTopic Method

Create a topic identity given a Topic name. This facility is provided for the rare cases where clients need to dynamically manipulate topic identity. This allows the creation of a topic identity with a provider specific name. Clients that depend on this ability are not portable.

Note: *This method is not for creating the physical topic. The physical creation of topics is an administration task and not to be initiated by the JMS interface. The one exception is the creation of temporary topics is done using the createTemporaryTopic method.*

```
public Topic createTopic(java.lang.String topicName)
   throws JMSException
```

Name	Description
topicName	The name of this topic.

Throws JMSException if a session fails to create a topic due to some JMS error.

The createSubscriber Method

Creates a non-durable Subscriber to the specified topic. A client uses a TopicSubscriber for receiving messages that have been published to a topic. Regular TopicSubscriber's are not durable. They only receive messages that are published while they are active.

In some cases, a connection may both publish and subscribe to a topic. The subscriber NoLocal attribute allows a subscriber to inhibit the delivery of messages published by its own connection. The default value for this attribute is false.

```
public TopicSubscriber createSubscriber(Topic topic)
   throws JMSException
```

Name	Description
topic	The topic to which to subscribe.

Throws JMSException if a session fails to create a subscriber due to some JMS error.

Throws InvalidDestinationException if invalid Topic specified.

The createSubscriber Method

Create a non-durable Subscriber to the specified topic. A client uses a TopicSubscriber for receiving messages that have been published to a topic. Regular TopicSubscriber's are not durable. They only receive messages that are published while they are active.

Messages filtered out by a subscriber's message selector will never be delivered to the subscriber. From the subscriber's perspective they simply don't exist. In some cases, a connection may both publish and subscribe to a topic. The subscriber NoLocal attribute allows a subscriber to inhibit the delivery of messages published by its own connection.

```
public TopicSubscriber createSubscriber(Topic topic,java.lang.String
messageSelector, boolean noLocal)
   throws JMSException
```

Name	Description
topic	The topic to which to subscribe.
messageSelector	Only messages with properties matching the message selector expression are delivered. This value may be null.
noLocal	If set, inhibits the delivery of messages published by it's own connection.

Throws JMSException if a session fails to create a subscriber due to some JMS error or invalid selector.

Throws InvalidDestinationException if invalid Topic specified.

Throws InvalidSelectorException if the message selector is invalid.

The createDurableSubscriber Method

Create a durable Subscriber to the specified topic. If a client needs to receive all the messages published on a topic including the ones published while the subscriber is inactive, it uses a durable TopicSubscriber. JMS retains a record of this durable subscription and insures that all messages from the topic's publishers are retained until they are either acknowledged by this durable subscriber or they have expired.

Sessions with durable subscribers must always provide the same client identifier. In addition, each client must specify a name which uniquely identifies (within client identifier) each durable subscription it creates. Only one session at a time can have a TopicSubscriber for a particular durable subscription.

A client can change an existing durable subscription by creating a durable TopicSubscriber with the same name and a new topic and/or message selector. Changing a durable subscriber is equivalent to unsubscribing(deleting) the old one and creating a new one.

```
public TopicSubscriber createDurableSubscriber(Topic topic,
java.lang.String name)
   throws JMSException
```

Name	Description
topic	The non-temporary topic to which to subscribe.
name	The name used to identify this subscription.

Throws JMSException if a session fails to create a subscriber due to some JMS error.

Throws InvalidDestinationException if invalid Topic specified.

The createDurableSubscriber Method

Create a durable Subscriber to the specified topic.

If a client needs to receive all the messages published on a topic including the ones published while the subscriber is inactive, it uses a durable TopicSubscriber. JMS retains a record of this durable subscription and insures that all messages from the topic's publishers are retained until they are either acknowledged by this durable subscriber or they have expired.

Sessions with durable subscribers must always provide the same client identifier. In addition, each client must specify a name which uniquely identifies (within client identifier) each durable subscription it creates. Only one session at a time can have a TopicSubscriber for a particular durable subscription. An inactive durable subscriber is one that exists but does not currently have a message consumer associated with it.

A client can change an existing durable subscription by creating a durable TopicSubscriber with the same name and a new topic and/or message selector. Changing a durable subscriber is equivalent to unsubscribing(deleting) the old one and creating a new one.

```
public TopicSubscriber createDurableSubscriber(Topic topic,
java.lang.String name, java.lang.String messageSelector, boolean
noLocal)
   throws JMSException
```

Name	Description
topic	Then name of the non-temporary topic to which to subscribe.
name	The name used to identify this subscription.
messageSelector	Only messages with properties matching the message selector expression are delivered. This value may be null.
noLocal	If set, inhibits the delivery of messages published by it's own connection.

Throws JMSException if a session fails to create a subscriber due to some JMS error or invalid selector.

Throws InvalidDestinationException if invalid Topic specified.

Throws InvalidSelectorException if the message selector is invalid.

The createPublisherMethod

Create a Publisher for the specified topic. A client uses a TopicPublisher for publishing messages on a topic. Each time a client creates a TopicPublisher on a topic, it defines a new sequence of messages that have no ordering relationship with the messages it has previously sent.

```
public TopicPublisher createPublisher(Topic topic)
   throws JMSException
```

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

Throws JMSException if a session fails to create a publisher due to some JMS error.

Throws InvalidDestinationException if invalid Topic specified.

The createTemporaryTopic Method

Create a temporary topic. It's lifetime will be that of the TopicConnection unless deleted earlier.

Throws JMSException if a session fails to create a temporary topic due to some JMS error.

```
public TemporaryTopic createTemporaryTopic()
   throws JMSException
```

The unsubscribe Method

Unsubscribe a durable subscription that has been created by a client. This deletes the state being maintained on behalf of the subscriber by its provider. It is erroneous for a client to delete a durable subscription while it has an active TopicSubscriber for it, or while a message received by it is part of a transaction or has not been acknowledged in the session.

```
public void unsubscribe(java.lang.String name)
   throws JMSException
```

Name	Description
name	The name used to identify this subscription

Throws JMSException if JMS fails to unsubscribe to durable subscription due to some JMS error.

Throws InvalidDestinationException if an invalid subscription name is specified.

7.3.36 interface javax.jms.XASession

```
public interface XASession
extends Session
```

The XASession interface extends the capability of Session by adding access to a JMS provider's support for the Java Transaction API (JTA) (optional). This support takes the form of a javax.transaction.xa.XAResource object. The functionality of this object closely resembles that defined by the standard X/Open XA Resource interface.

An application server controls the transactional assignment of an XASession by obtaining its XAResource. It uses the XAResource to assign the session to a transaction; prepare and commit work on the transaction; etc.

An XAResource provides some fairly sophisticated facilities for interleaving work on multiple transactions; recovering a list of transactions in progress; etc. A JTA aware JMS provider must fully implement this functionality. This could be done by using the services of a database that supports XA or a JMS provider may choose to implement this functionality from scratch.

A client of the application server is given what it thinks is a regular JMS Session. Behind the scenes, the application server controls the transaction management of the underlying XASession.

The getXAResource Method

Returns an XA resource to the caller.

```
public javax.transaction.xa.XAResource getXAResource()
```

The getTransacted Method

Queries whether the session is in transacted mode.

```
public boolean getTransacted()
   throws JMSException
```

Throws JMSException if JMS fails to return the transaction mode due to internal error in JMS Provider.

Specified by getTransacted in interface Session

Throws JMSException if JMS fails to return the transaction mode due to internal error in JMS Provider.

The commit Method

Throws a TransactionInProgressException, since it should not be called for an XASession object.

```
public void commit()
   throws JMSException
```

Throws TransactionInProgressException if method is called on a XASession.

Specified by commit in interface Session

Throws TransactionInProgressException if method is called on a XASession.

The rollback Method

Throws a TransactionInProgressException, since it should not be called for an XASession object.

```
public void rollback()
   throws JMSException
```

Specified by rollback in interface Session

Throws TransactionInProgressException if method is called on a XASession.

7.3.37 interface javax.jms.XAQueueSession

```
public interface XAQueueSession
extends XASession
```

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

The getQueueSession Method

Gets the queue session associated with this XAQueueSession.

```
public QueueSession getQueueSession()
   throws JMSException
```

Throws JMSException if a JMS error occurs.

7.3.38 interface javax.jms.XATopicSession

```
public interface XATopicSession
extends XASession
```

An XATopicSession provides a regular TopicSession. which can be used to create TopicSubscriber and TopicPublisher objects (optional).

getTopicSession

```
public TopicSession getTopicSession()
   throws JMSException
```

Gets the topic session associated with this XATopicSession.

Throws JMSException - if a JMS error occurs.

7.3.39 interface javax.jms.XAConnection

```
public interface XAConnection
```

XAConnection extends the capability of Connection by providing an XASession (optional).

7.3.40 interface javax.jms.XAQueueConnection

```
public interface XAQueueConnection
extends XAConnection, QueueConnection
```

XAQueueConnection provides the same create options as QueueConnection (optional). The only difference is that an XAConnection is by definition transacted.

The createXAQueueSession Method

Creates an XAQueueSession.

```
public XAQueueSession createXAQueueSession()
   throws JMSException
```

Throws JMSException if JMS Connection fails to create a XA queue session due to some internal error.

The createQueueSession Method

Creates an XAQueueSession.

```
    public QueueSession createQueueSession(boolean transacted, int
    acknowledgeMode)
    throws JMSException
```

Specified by createQueueSession in interface QueueConnection

Name	Description
transacted	ignored.
acknowledgeMode	ingnored.

Throws JMSException if JMS Connection fails to create a XA queue session due to some internal error.

7.3.41 interface javax.jms.XATopicConnection

An XATopicConnection provides the same create options as TopicConnection (optional). The only difference is that an XAConnection is by definition transacted.

The createXATopicSession Method

Creates an XATopicSession.

```
public XATopicSession createXATopicSession()
   throws JMSException
```

Throws JMSException if JMS Connection fails to create a XA topic session due to some internal error.

The createTopicSession Method

Creates an XATopicSession

```
public TopicSession createTopicSession(boolean transacted, int
acknowledgeMode)
   throws JMSException
```

Name	Description
transacted	ignored.
acknowledgeMode	ingnored.

Specified by createTopicSession in interface TopicConnection

Throws JMSException if JMS Connection fails to create a XA topic session due to some internal error.

7.3.42 interface javax.jms.XAConnectionFactory

```
public interface XAConnectionFactory
```

The XAConnectionFactory interface is a base interface for the XAQueueConnectionFactory and XATopicConnectionFactory interfaces.

7.3.43 interface javax.jms.XAQueueConnectionFactory

```
public interface XAQueueConnectionFactory
extends XAConnectionFactory, QueueConnectionFactory
```

An XAQueueConnectionFactory provides the same create options as a QueueConnectionFactory (optional).

The createXAQueueConnection Method

Creates an XA queue connection with default user identity. The connection is created in stopped mode. No messages will be delivered until Connection.start method is explicitly called.

```
public XAQueueConnection createXAQueueConnection()
   throws JMSException
```

Throws JMSException if JMS Provider fails to create XA queue Connection due to some internal error.

Throws JMSSecurityException if client authentication fails due to invalid user name or password.

The createXAQueueConnection Method

Creates an XA queue connection with specific user identity. The connection is created in stopped mode. No messages will be delivered until `Connection.start` method is explicitly called.

```
public XAQueueConnection createXAQueueConnection(java.lang.String
userName, java.lang.String password)
   throws JMSException
```

Name	Description
userName	The caller’s username
password	The caller’s password.

Throws JMSException if JMS Provider fails to create XA queue Connection due to some internal error.

Throws JMSSecurityException if client authentication fails due to invalid user name or password.

7.3.44 interface javax.jms.XATopicConnectionFactory

```
public interface XATopicConnectionFactory
```

An XATopicConnectionFactory provides the same create options as a TopicConnectionFactory (optional).

7.4 Unsupported Java JMS Classes

- class javax.jms.QueueRequestor
- class javax.jms.TopicRequestor

7.5 Unsupported Java JMS Interfaces

- interface javax.jms.ConnectionConsumer
- interface javax.jms.QueueBrowser
- interface javax.jms.ServerSession
- interface javax.jms.ServerSessionPool

7.6 Unsupported JMS Methods

Of the classes that are currently supported, the following methods are NOT supported:

- Interface QueueConnection
  - createConnectionConsumer(Queue queue, java.lang.String messageSelector, ServerSessionPool sessionPool, int maxMessages
- Interface TopicConnection
  - createConnectionConsumer(Topic topic, java.lang.String messageSelector, ServerSessionPool sessionPool, int maxMessages)
  - createDurableConnectionConsumer(Topic topic, java.lang.String subscriptionName, java.lang.String messageSelector, ServerSessionPool sessionPool, int maxMessages)
- Interface TopicSubscriber
  - createBrowser(Queue queue)
  - createBrowser(Queue queue, java.lang.String messageSelector)
- Interface TopicSubscriber
  - getNoLocal()
- Interface XAResource
  - forget(Xid xid)

7.7 The Message Service COM+ APIs

7.7.1 Supported Java Message Service (JMS) Classes for COM+

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

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

You may also find useful the following books:

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

7.7.2 The BytesMessage Object

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

Methods of the BytesMessage Object

The Acknowledge Method

Acknowledges the receipt of current and previous messages.

```
BytesMessage.acknowledge
```

The ClearBody Method

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

```
BytesMessage.ClearBody
```

The ClearProperties Method

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

```
BytesMessage.ClearProperties
```

The GetProperty Method

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

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

Name	Description
name	The name of the property.

The PropertyExists Method

Checks whether a value for a specific property exists.

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

Name	Description
name	The name of the property to check.

The ReadBoolean Method

Reads a Boolean value from the bytes message stream.

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

The ReadByte Method

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

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

The ReadBytes Method

Reads a portion of the bytes message stream.

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

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

The ReadChar Method

Reads a Unicode character value from the bytes message stream.

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

The ReadDouble Method

Reads a double from the bytes message stream.

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

The ReadFloat Method

Reads a float from the bytes message stream.

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

The ReadInt Method

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

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

The ReadLong Method

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

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

The ReadShort Method

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

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

The ReadUnsignedByte Method

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

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

The ReadUnsignedShort Method

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

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

The ReadUTF Method

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

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

The Reset Method

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

```
BytesMessage.Reset
```

The SetProperty Method

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

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

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

The WriteBoolean Method

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

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

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

The WriteByte Method

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

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

Name	Description
value	The byte value to be written

The WriteBytes Method

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

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

Name	Description
value	The byte array value to be written.

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

The WriteChar Method

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

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

Name	Description
value	The Char value to be written.

The WriteDouble Method

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

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

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

The WriteFloat Method

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

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

Name	Description
value	The float value to be written.

The WriteInt Method

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

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

Name	Description
value	The float value to be written.

The WriteLong Method

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

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

Name	Description
value	The WriteLong is written as a currency.

The WriteObject Method

Currently not supported

The WriteShort Method

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

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

Name	Description
value	The short that is written.

The WriteUTF Method

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

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

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

Properties of the BytesMessage Object

The CorrelationID Property

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

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

The CorrelationIDAsBytes Property

Currently not supported.

The DeliveryMode Property

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

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

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

The Destination Property

Currently not supported.

The Expiration Property

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

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

The MessageID Property

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

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

The Priority Property

```
Currently not supported.
```

The Redelivered Property

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

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

The ReplyTo Property

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

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

The Timestamp Property

The TimeStamp property sets or returns the message timestamp.

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

The Type Property

The Type property sets or returns the message type.

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

7.7.3 The Connection Object

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

Methods of the Connection Object

The Start Method

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

```
Connection.Start
```

The Stop Method

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

```
Connection.Stop
```

Properties of the Connection Object

The ClientID Property

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

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

The MetaData Property

This property is not currently supported.

7.7.4 The ConnectionFactory Object

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

Methods of the ConnectionFactory Object

There are no methods currently associated with this object.

Properties of the ConnectionFactory Object

The HostName Property

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

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

The Port Property

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

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

The PortOffset Property

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

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

7.7.5 The Connection MetaData Object

This Object is currently not supported.

7.7.6 The MapMessage Object

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

Methods of the MapMessage Object

The Acknowledge Method

Acknowledges the receipt of current and previous messages.

```
MapMessage.Acknowledge
```

The ClearBody Method

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

```
MapMessage.ClearBody
```

The ClearProperties Method

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

```
MapMessage.ClearProperties
```

The GetBoolean Method

The GetBoolean method returns the boolean value with the given name

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

Name	Description
name	The name of the Boolean property.

The GetByte Method

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

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

Name	Description
name	The name of the byte.

The GetBytes Methods

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

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

Name	Description
name	The name of the byte property.

Name	Description
length	The length of the property

The GetChar Method

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

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

Name	Description
name	The name of the Unicode character.

The GetDouble Method

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

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

Name	Description
name	The name of the double property.

The GetFloat Method

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

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

Name	Description
name	The name of the float property.

The GetInt Method

The GetInt method returns the long value with the given name

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

Name	Description
name	The name of the integer.

The GetLong Method

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

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

Name	Description
name	The name of the currency property.

The GetObject Method

The GetObject method is currently not supported.

The GetProperty Method

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

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

Name	Description
name	Name of the currency property.

The GetShort Method

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

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

Name	Description
name	The name of the short currency property.

The GetString Method

Return the String value with the given name

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

Name	Description
name	The name of the String property.

The ItemExists Method

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

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

Name	Description
name	The name of the item to check.

The PropertyExists Method

The PropertyExists method checks if a property value exists.

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

Name	Description
name	The name of the property value.

The SetBoolean Method

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

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

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

The SetByte Method

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

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

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

The SetBytes Method

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

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

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

The SetChar Method

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

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

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

The SetDouble Method

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

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

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

The SetFloat Methods

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

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

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

The SetInt Method

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

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

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

The SetLong Method

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

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

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

The SetObject Method

This method is currently not supported.

The SetProperty Method

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

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

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

The SetShort Method

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

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

Name	Description
name	The name of the short property.

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

The SetString Method

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

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

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

Properties of the MapMessage Object

The CorrelationID Property

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

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

The CorrelationIDAsBytes Property

Currently not supported.

The DeliveryMode Property

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

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

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

The Destination Property

Currently not supported.

The Expiration Property

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

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

The MapNames Property

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

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

The MessageID Property

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

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

The Priority Property

Currently not supported.

The Redelivered Property

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

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

The ReplyTo Property

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

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

The Timestamp Property

The TimeStamp property sets or returns the message timestamp.

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

The Type Property

The Type property sets or returns the message type.

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

7.7.7 The Message Object

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

Methods of the Message Object

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

The Acknowledge Method

Acknowledges the receipt of current and previous messages.

```
Message.acknowledge
```

The ClearBody Method

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

```
Message.ClearBody
```

The ClearProperties Method

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

```
Message.ClearProperties
```

The GetProperty Method

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

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

Name	Description
name	The name of the property.

The PropertyExists Method

Checks whether a value for a specific property exists.

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

Name	Description
name	The name of the property to check.

The SetProperty Method

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

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

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

Properties of the Message Object

The CorrelationID Property

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

```
Message.CorrelationID = String
```

```
String = Message.CorrelationID
```

The CorrelationIDAsBytes Property

The CorrelationIDAsBytes is not currently supported.

The DeliveryMode Property

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

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

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

The Destination Property

Currently not supported.

The Expiration Property

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

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

The MessageID Property

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

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

The Priority Property

Currently not supported.

The Redelivered Property

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

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

The ReplyTo Property

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

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

The Timestamp Property

The TimeStamp property sets or returns the message timestamp.

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

The Type Property

The Type property sets or returns the message type.

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

7.7.8 The MessageConsumer Object

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

Methods of the MessageConsumer Object

The Close Method

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

```
MessageConsumer.Close
```

The Receive Message Method

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

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

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

The ReceiveNoWait Method

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

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

Properties of the MessageConsumer Object

The MessageListener Property

This property is currently not supported.

The MessageSelector Property

The MessageSelector propert returns this message consumer’s message selector expression.

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

7.7.9 The MessageListener Object

This object is currently not supported.

The OnMessage Property

This function is currently not supported.

7.7.10 The MessageProducer Object

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

Methods of the MessageProducer Object

There are no methods associated with this object.

Properties of the MessageProducer Object

The DeliveryMode Property

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

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

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

The DisableMessageID Property

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

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

The DisableMessageTimestamp Property

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

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

The Priority Method

Currently not supported.

The TimeToLive Method

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

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

7.7.11 The Queue Object

A Queue object encapsulates a Message Server specific queue name.

Methods of the Queue Object

The ToString Method

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

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

Properties of the Queue Object

The QueueName Property

Returns the name of this queue. Read-only.

7.7.12 The QueueBrowser Object

This object is currently not supported.

7.7.13 The QueueConnection Object

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

Methods of the QueueConnection Object

The CreateQueueSession Method

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

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

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

The Start Method

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

```
QueueConnection.Start
```

The Stop Method

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

```
QueueConnection.Stop
```

Properties of QueueConnection Object

The ClientID Property

Returns or sets client identifier for this connection.

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

The MetaData Property

Not currently supported.

7.7.14 The QueueConnectionFactory Object

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

Methods of the QueueConnectionFactory Object

The CreateQueueConnection Method

Create a queue connection with a default user identity.

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

Properties of the QueueConnectionFactory Object

The HostName Property

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

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

The Port Property

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

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

The PortOffset Property

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

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

7.7.15 The QueueReceiver Object

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

Methods of the QueueReceiver Object

The Close Method

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

```
QueueReceiver.Close
```

The Receive Method

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

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

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

The ReceiveNoWait Method

Receive the next message if one is immediately available.

```
QueueReceiver.ReceiveNoWait As message
```

Properties of the QueueReceiver Object

The MessageListener Property

This property is not currently supported.

The MessageSelector Property

Returns this message consumer's message selector expression.

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

The Queue Property

Returns the queue associated with this queue receiver.

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

7.7.16 The QueueRequestor Object

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

Methods of the QueueRequestor Object

The Create Method

Constructs the QueueRequestor.

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

Name	Description
session	The QueueSession.
queue	Queue name.

The Request Method

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

QueueRequestor.Request(*message As message*) As message

Name	Description
message	The message.

7.7.17 The QueueSender Object

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

Methods of the QueueSender Object

The Send Method

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

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

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

Properties of the QueueSender Object

The DeliveryMode Property

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

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

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

The DisableMessageID Property

Returns or sets an indication of whether message IDs are disabled

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

The DisableMessageTimestamp Property

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

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

The Priority Property

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

The Queue Property

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

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

The TimeToLive Property

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

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

7.7.18 The QueueSession Object

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

Methods of the QueueSession Object

The Commit Method

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

```
QueueSession.Commit
```

The CreateBrowser Method

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

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

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

The CreateBytesMessage Method

Create a BytesMessage.

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

The CreateMapMessage Method

Create a MapMessage.

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

The CreateMessage Method

Create a Message.

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

The CreateQueue Method

Create a queue identity given a Queue name.

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

Name	Description
QueueName	The name of the queue.

The CreateReceiver Method

Create a QueueReceiver to receive messages for the specified queue.

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

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

The CreateSender Method

Create a QueueSender to send messages to the specified queue..

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

Name	Description
Queue	The name of the queue.

The CreateStreamMessage Method

Create a StreamMessage.

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

The CreateTemporaryQueue Method

Create a temporary queue.

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

The CreateTextMessage Method

Create aTextMessage.

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

Name	Description
Text	The string used to initialize this message.

The Recover Method

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

```
QueueSession.Recover()
```

The Rollback Method

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

```
QueueSession.Rollback()
```

The Run Method

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

```
QueueSession.Run()
```

Properties of the QueueSender Object

The MessageListener Property

This property is not currently supported.

The Transacted Property

Returns an indication that the session is in transacted mode.

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

7.7.19 The Session Object

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

Methods of the Session Object

The Commit Method

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

```
Session.Commit
```

The CreateBytesMessage Method

The CreateBytesMessage method creates a BytesMessage.

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

The CreateMapMessage Method

The CreateMapMessage method creates a MapMessage.

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

The CreateMessage Method

Create a Message.

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

The CreateStreamMessage Method

Create a StreamMessage.

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

The CreateTextMessage Method

Create a TextMessage.

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

Name	Description
Text	The string used to initialize this message.

The Recover Method

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

```
Session.Recover
```

The Rollback Method

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

```
Session.Rollback
```

The Run Method

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

```
Session.Run
```

Properties of the Session Object

The MessageListener Property

This property is currently not supported.

The Transacted Property

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

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

7.7.20 The StreamMessage Object

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

The Acknowledge Method

Acknowledges the receipt of current and previous messages.

```
StreamMessage.acknowledge
```

The ClearBody Method

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

```
StreamMessage.ClearBody
```

The ClearProperties Method

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

```
StreamMessage.ClearProperties
```

The GetProperty Method

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

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

Name	Description
name	The name of the property.

The PropertyExists Method

Checks whether a value for a specific property exists.

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

Name	Description
name	The name of the property to check.

The ReadBoolean Method

Reads a Boolean value from the bytes message stream.

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

The ReadByte Method

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

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

The ReadBytes Method

Reads a portion of the bytes message stream.

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

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

The ReadChar Method

Reads a Unicode character value from the bytes message stream.

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

The ReadDouble Method

Reads a double from the bytes message stream.

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

The ReadFloat Method

Reads a float from the bytes message stream.

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

The ReadInt Method

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

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

The ReadLong Method

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

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

The ReadObject Method

Currently not supported.

The ReadShort Method

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

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

The ReadString Method

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

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

The Reset Method

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

```
StreamMessage.Reset
```

The SetProperty Method

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

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

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

The WriteBoolean Method

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

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

Name	Description
value	The boolean value to be written.

The WriteByte Method

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

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

Name	Description
value	The byte value to be written.

The WriteBytes Method

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

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

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

The WriteChar Method

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

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

Name	Description
value	The char value to be written.

The WriteDouble Method

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

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

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

The WriteFloat Method

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

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

Name	Description
value	The float value to be written.

The WriteInt Method

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

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

Name	Description
value	The int value to be written.

The WriteLong Method

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

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

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

The WriteObject Method

Currently not supported

The WriteShort Method

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

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

Name	Description
value	The short that is written.

The WriteString Method

Write a string to the message stream.

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

Name	Description
value	The String value that is written.

Properties of the StreamMessage Object

The CorrelationID Property

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

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

The CorrelationIDAsBytes Property

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

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

The DeliveryMode Property

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

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

Name	Description
msDefaultDeliveryMode	Default DeliveryMode delivery mode.

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

The Destination Property

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

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

The Expiration Property

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

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

The MessageID Property

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

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

The Priority Property

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

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

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

The Redelivered Property

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

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

The ReplyTo Property

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

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

The Timestamp Property

The TimeStamp property sets or returns the message timestamp.

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

The Type Property

The Type property sets or returns the message type.

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

7.7.21 The TemporaryQueue Object

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

Methods of the TemporaryQueue Object

The Delete Method

The Delete method deletes the temporary queue.

```
TemporaryQueue.Delete
```

The ToString Method

The ToString method returns a printed version of the queue name

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

Properties of the TemporaryQueue Object

The QueueName Property

The QueueName property returns the name of this queue.

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

7.7.22 The TemporaryTopic Object

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

Methods of the TemporaryTopic Object

The Delete Method

The Delete method deletes the temporary topic.

```
TemporaryTopic.Delete
```

The ToString Method

The ToString method returns a printed version of the topic name

```
TemporaryTopic.ToString
```

Properties of the TemporaryTopic Object

The TopicName Property

The TopicName property returns the name of this topic.

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

7.7.23 The TextMessage Object

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

Methods of the TextMessage Object

The Acknowledge Method

Acknowledges the receipt of current and previous messages.

```
TextMessage.acknowledge
```

The ClearBody Method

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

```
TextMessage.ClearBody
```

The ClearProperties Method

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

```
TextMessage.ClearProperties
```

The GetProperty Method

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

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

Name	Description
name	The name of the property.

The PropertyExists Method

Checks whether a value for a specific property exists.

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

Name	Description
name	The name of the property to check.

The SetProperty Method

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

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

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

Properties of the Message Object

The CorrelationID Property

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

```
TextMessage.CorrelationID = String
String = TextMessage.CorrelationID
```

The CorrelationIDAsBytes Property

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

```
Message.CorrelationIDAsBytes = Variant
Variant = Message.CorrelationIDAsBytes
```

The DeliveryMode Property

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

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

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

The Destination Property

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

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

The Expiration Property

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

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

The MessageID Property

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

```
TextMessage.MessageID = String
String = TextMessage.MessageID
```

The Priority Property

The Priority property sets or returns the priority that is assigned to this message. Possible numbers are 1 - 9. (Not currently supported, but suggested that the value be entered, to prevent code changes later.)

```
TextMessage.Priority = PriorityConstant
PriorityConstant = TextMessage.Priority
```

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

The Redelivered Property

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

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

The ReplyTo Property

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

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

The Text Property

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

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

The Timestamp Property

The TimeStamp property sets or returns the message timestamp.

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

The Type Property

The Type property sets or returns the message type.

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

7.7.24 The Topic Object

A Topic object encapsulates a Message Server specific topic name.

Methods of the Topic Object

The ToString Method

The ToString method returns a printed version of the topic name

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

Properties of the Topic Object

The TopicName Property

The TopicName property returns the name of this topic.

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

7.7.25 The TopicConnection Object

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

Methods of the TopicConnection Object

The CreateTopicSession Method

Create a TopicSession

```
TopicConnection.CreateTopicSession(Transacted As Boolean,
acknowledgeMode As AcknowledgeModeConstants) As TopicSession
```

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

The Start Method

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

```
TopicConnection.Start
```

The Stop Method

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

TopicConnection.Stop

Properties of the TopicConnection

The ClientID Property

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

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

The MetaData Property

This property is currently not supported.

7.7.26 The TopicConnectionFactory Object

A client uses a TopicconnectionFactory to create TopicConnections with a Pub/Sub Message Server.

Methods of the TopicConnectionFactory Object

The CreateTopicConnection Method

Create a topic connection with default user identity.

```
TopicConnectionFactory.CreateTopicConnection() As TopicConnection
```

Properties of the TopicConnectionFactory

The HostName Property

The HostName property sets or returns the host name of the machine that the JMS IQ Server is running on.

```
TopicConnectionFactory.HostName = String
String = TopicConnecitonFactory.HostName
```

The Port Property

The Port property sets or returns the port number that the JMS IQ Server is listening on, default value is 7555

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

The PortOffset Property

The PortOffset sets or returns the port offset number of the JMS IQ Server if more then one Message Server is running on same host machine and using same port number.

```
TopicConnectionFactory.PortOffset = Long
Long = TopicConnectionFactory
```

7.7.27 The TopicPublisher Object

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

Methods of the TopicPublisher Object

The Publish Method

The Publish method publishes a Message to a topic for an unidentified message producer, specifying delivery mode, priority and time to live.

```
TopicPublisher.Publish(message As message, [DeliveryMode],
[Priority], [TimeToLive], [Topic])
```

Name	Description
message	The message to publish.
deliveryMode	The delivery mode to use.
priority	The priority for this message. While not currectly supported, it is recommended to implement now, to prevent code changes later.
timeToLive	The message's lifetime (in milliseconds).
topic	The topic to publish this message to.

Properties of TopicPublisher

The DeliveryMode Property

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

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

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

The DisableMessageID Property

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

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

The DisableMessageTimestamp Property

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

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

The Priority Property

The Priority property sets or returns the priority that is assigned to this message. Possible numbers are 1 - 9. While not currently supported, it is suggested that the desired value be entered now, to prevent code changes later.

```
TopicPublisher.Priority = PriorityConstant
PriorityConstant = TopicPublisher.Priority
```

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

The TimeToLive Property

The TimeToLive property sets and returns the default length of time in milliseconds from its dispatch time that a produced message should be retained by the message system.

```
TopicPublisher.TimeToLive = MessageConstant
MessageConstant = TopicPublisher.TimeToLive
```

Name	Description
msDefaultTimeToLive	The default value of 0 = Unlimited

The Topic Property

The Topic property returns the topic associated with this publisher.

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

7.7.28 The TopicRequestor Object

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

Methods of the TopicRequestor Object

The Create Method

Constructs the TopicRequestor.

```
TopicRequestor.Create(session As TopicSession, Topic As Topic)
```

Name	Description
session	The name of the topic session.
topic	The name of the topic.

The Request Method

Send a request and wait for a reply

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

Name	Description
message	The message text.

7.7.29 The TopicSession Object

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

Methods of the TopicSession Object

The Commit Method

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

```
TopicSession.Commit
```

The CreateBytesMessage Method

The CreateBytesMessage method creates a BytesMessage.

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

The CreateDurableSubscriber Method

The CreateDurableSubscriber method creates a durable Subscriber to the specified topic.

```
TopicSession.CreateDurableSubscriber(Topic As Topic, name As String,
[MessageSelector], [NoLocal] As TopicSubscriber
```

Name	Description
`topic`	The non-temporary topic to subscribe to.
name	The name used to identify this subscription.
messageSelector	Only messages with properties matching the message selector expression are delivered. You may use a null.
noLocal	If set, noLocal inhibits the delivery of messages published by its own connection.

The CreateMapMessage Method

The CreateMapMessage method creates a MapMessage.

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

The CreateMessage Method

Create a Message.

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

The CreatePublisher Method

Create a Publisher for the specified topic.

```
TopicSession.CreatePublisher(Topic As Topic) As TopicPublisher
```

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

The CreateStreamMessage Method

Create a StreamMessage.

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

The CreateSubscriber Method

Create a non-durable Subscriber to the specified topic

```
TopicSession.CreateSubscriber(Topic As Topic, [MessageSelector],
[NoLocal]) As TopicSubscriber
```

Name	Description
`topic`	The topic to subscribe to.
`messageSelector`	Only messages with properties matching the message selector expression are delivered. This value may be null.
`noLocal`	If set, inhibits the delivery of messages published by its own connection.

The CreateTemporaryTopic Method

The CreateTemporaryTopic method creates a temporary topic.

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

The CreateTextMessage Method

The CreateTextMessage method creates TextMessage.

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

Name	Description
`text`	The string used to initialize this message.

The CreateTopic Method

Create a topic identity given a Topic name.

```
TopicSession.CreateTopic(TopicName As String) As Topic
```

Name	Description
`topicName`	The name of this topic.

The Recover Method

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

```
TopicSession.Recover
```

The Rollback Method

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

```
TopicSession.Rollback
```

The Run Method

The Run method is an optional method

```
TopicSession.Run
```

The Unsubscribe Method

The Unsubscribe method unsubscribes a durable subscription that has been created by a client.

```
TopicSession.Unsubscribe(name As String)
```

Name	Description
name	The name used to identify this subscription.

Properties of the TopicSession Object

The MessageListener Property

This property is currently not supported.

The Transacted Property

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

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

7.7.30 The TopicSubscriber Object

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

Methods of the TopicSubscriber Object

The Close Method

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

```
TopicSubscriber.Close
```

The Receive Method

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

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

Name	Description
timeout	The timeout value (in milliseconds).

The ReceiveNoWait Method

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

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

Properties of the TopicSubscriber Object

The MessageListener Property

This property is currently not supported.

The MessageSelector Property

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

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

The NoLocal Property

The NoLocal property returns the NoLocal attribute for this TopicSubscriber.

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

The Topic Property

The Topic property returns the topic associated with this subscriber.

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

7.7.31 The XAQueueConnection Object

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

Methods of the XAQueueConnection Object

The CreateQueueSession Method

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

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

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

The CreateXAQueueSession Method

Create an XAQueueSession.

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

The Start Method

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

```
XAQueueConnection.Start
```

The Stop Method

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

```
XAQueueConnection.Stop
```

Properties of XAQueueConnection Object

The ClientID Property

Returns or sets client identifier for this connection.

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

The MetaData Property

Not currently supported.

7.7.32 The XAQueueConnectionFactory Object

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

Methods of the XAQueueConnectionFactory Object

The CreateQueueConnection Method

Create a queue connection with a default user identity.

```
XAQueueConnectionFactory.CreateQueueConnection() As QueueConnection
```

The CreateXAQueueConnection Method

Create an XA queue connection with a default user identity.

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

Properties of the QueueConnectionFactory Object

The HostName Property

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

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

The Port Property

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

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

The PortOffset Property

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

```
XAQueueConnectionFactory.PortOffset = Long
Long = XAQueueConnectionFactory.PortOffset
```

7.7.33 The XAQueueSession Object

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

Methods of the QueueSession Object

The Commit Method

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

```
XAQueueSession.Commit
```

The CreateBytesMessage Method

Create a BytesMessage.

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

The CreateMapMessage Method

Create a MapMessage.

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

The CreateMessage Method

Create a Message.

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

The CreateStreamMessage Method

Create a StreamMessage.

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

The CreateTextMessage Method

Create aTextMessage.

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

Name	Description
Text	The string used to initialize this message.

The Recover Method

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

```
XAQueueSession.Recover()
```

The Rollback Method

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

```
XAQueueSession.Rollback()
```

The Run Method

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

```
XAQueueSession.Run()
```

Properties of the QueueSender Object

The MessageListener Property

This property is not currently supported.

The QueueSession Property

Returns the queue session associated with this XAQueueSession.

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

The Transacted Property

Returns an indication that the session is in transacted mode.

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

7.7.34 The XASession Object

The XASession extends the capability of Session by adding access to a Message Server's support for Transaction, using the Compensating Resource Manager (CRM), handled under the Distributed Transaction Coordinator (DTC).

Methods of the Session Object

The Commit Method

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

```
XASession.Commit
```

The CreateBytesMessage Method

The CreateBytesMessage method creates a BytesMessage.

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

The CreateMapMessage Method

The CreateMapMessage method creates a MapMessage.

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

The CreateMessage Method

Create a Message.

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

The CreateStreamMessage Method

Create a StreamMessage.

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

The CreateTextMessage Method

Create a TextMessage.

`XASession.CreateTextMessage([`*`Text`*`])`

Name	Description
Text	The string used to initialize this message.

The Recover Method

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

```
XASession.Recover
```

The Rollback Method

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

```
XASession.Rollback
```

The Run Method

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

```
XASession.Run
```

Properties of the Session Object

The MessageListener Property

This property is currently not supported.

The Transacted Property

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

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

7.7.35 The XATopicConnection Object

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

Methods of the TopicConnection Object

The CreateTopicSession Method

Create a TopicSession

```
XATopicConnection.CreateTopicSession(Transacted As Boolean,
acknowledgeMode As AcknowledgeModeConstants) As TopicSession
```

Name	Description
Transacted	If true, session is transacted.

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

The Start Method

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

```
XATopicConnection.Start
```

The Stop Method

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

```
XATopicConnection.Stop
```

Properties of the TopicConnection

The ClientID Property

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

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

The MetaData Property

This property is currently not supported.

7.7.36 The XATopicConnectionFactory Object

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

Methods of the TopicConnectionFactory Object

The CreateTopicConnection Method

Create a topic connection with default user identity.

```
XATopicConnectionFactory.CreateTopicConnection() As TopicConnection
```

The CreateXATopicConnection Method

Create an XA topic connection with default user identity.

```
XATopicConnectionFactory.CreateTopicConnection() As TopicConnection
```

Properties of the TopicConnectionFactory

The HostName Property

The HostName property sets or returns the host name of the machine that the JMS IQ Server is running on.

```
XATopicConnectionFactory.HostName = String
String = XATopicConnecitonFactory.HostName
```

The Port Property

The Port property sets or returns the port number that the JMS IQ Server is listening on, default value is 7555

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

The PortOffset Property

The PortOffset sets or returns the port offset number of the JMS IQ Server if more then one Message Server is running on same host machine and using same port number.

```
XATopicConnectionFactory.PortOffset = Long
Long = XATopicConnectionFactory
```

7.7.37 The XATopicSession Object

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

Methods of the XATopicSession Object

The Commit Method

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

```
XATopicSession.Commit
```

The CreateBytesMessage Method

The CreateBytesMessage method creates a BytesMessage.

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

The CreateMapMessage Method

The CreateMapMessage method creates a MapMessage.

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

The CreateMessage Method

Create a Message.

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

The CreateStreamMessage Method

Create a StreamMessage.

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

The CreateTextMessage Method

The CreateTextMessage method creates TextMessage.

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

Name	Description
`text`	The string used to initialize this message.

The Recover Method

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

```
XATopicSession.Recover
```

The Rollback Method

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

```
TopicSession.Rollback
```

The Run Method

The Run method is an optional method

```
TopicSession.Run
```

Properties of the TopicSession Object

The MessageListener Property

This property is currently not supported.

The TopicSession Property

Returns the topic session associated with this XATopicSession.

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

The Transacted Property

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

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

Chapter 8

Client Libraries for the Multiplexer e*Way

The e*Gate API Kit Multiplexer e*Way contains the following types of function sets:

- **C API Function Prototypes** on page 251
- **ActiveX APIs** on page 260
- **Java Methods** on page 265
  - **com.stc.MUXPooler** on page 275
- **Perl Subroutines** on page 282

8.1 C API Function Prototypes

The file **ewipmpclnt.h** defines the following function prototypes:

- **EWIPMP_Close** on page 251
- **EWIPMP_Free** on page 252
- **EWIPMP_Open** on page 253
- **EWIPMP_Send** on page 254
- **EWIPMP_Wait** on page 254

Note: *The comments within the ewipmpclnt.h header file contain the same information as this chapter.*

EWIPMP_Close

Syntax

```
EWIPMP_Close(hIPMP)
```

Description

EWIPMP_Close closes an IPMP connection and frees all resources associated with the HEWIPMP handle.

Parameters

Name	Type	Description
hIPMP	HEWIPMP	An IPMP connection handle.

Return Values

Boolean

Returns **true** if the connection was successfully closed; otherwise, returns **false**. Use **getlasterror()** to obtain any error codes.

Examples

```
//close connection handle
if(hIPMP)
{
    EWIPMP_Close(hIPMP)
}
```

EWIPMP_Free

Syntax

EWIPMP_Free(*hIPMP,pbReturnMessage*)

Description

EWIPMP_Free frees the memory associated with the pbReturnMessage in the EWIMP_Wait call.

Parameters

Name	Type	Description
hIPMP	HEWIPMP	An IPMP connection handle.
pbReturnMessage	byte *	A message, as a blob.

Return Values

Returns **true** if the operation was performed successfully; otherwise, returns **false**. Use **getlasterror()** to obtain any error codes.

Examples

```
// free message buffer
    //
    if (!EWIPMP_Free(hIPMP, (BYTE*)pBuffer))
    {
        pBuffer = NULL;
        goto FreeFailed;
    }
```

EWIPMP_Open

Syntax

```
EWIPMP_Open(phIPMP,pcszServerHost,dwServerPort,dwflags,
            pvReserved)
```

Description

EWIPMP_Open creates and initializes an IPMP connection with a remote host.

Parameters

Name	Type	Optional	Description
phIPMP	pointer		A pointer to the connection handle. Pass this function an address of an empty handle, and the function returns the handle for use by other IPMP functions.
pcszServerHost	char	yes	The name of the remote host. If this parameter is null, "localhost" is used.
dwServerPort	dword	yes	The TCP/IP port number. If this parameter is zero, the default 26051 is used.
dwflags	dword		Reserved for future use and ***must*** be set to zero.
pvReserved	void		Reserved for future use and ***must*** be set to null.

Return Values

Boolean

Returns **true** and creates an IPMP connection handle if the connection was successfully established; otherwise, returns **false**. Use **getlasterror()** to obtain any error messages.

Additional Notes

The caller must call the EWIMP_Close method (See **EWIPMP_Close** on page 251) to release the connection and any resources associated with the handle.

Examples

```
// open connection to the multiplexer e*way
    //

    if (pszServerHost == NULL)
    {
        if (!EWIPMP_Open(&hIPMP, "localhost", dwServerPort, 0, NULL))
        {
            goto OpenConnectionFailed;
        }
    }
    else
    {
        if (!EWIPMP_Open(&hIPMP, pszServerHost, dwServerPort, 0, NULL))
        {
            goto OpenConnectionFailed;
        }
    }
```

EWIPMP_Send

Syntax

```
EWIPMP_Send(hIPMP,cbMessage,pbMessage,cSecondsToExpire,dwflags,
pvReserved)
```

Description

EWIPMP_Send sends the specified message, optionally with the specified expiration time.

Parameters

Name	Type	Optional	Description
hIPMP	handle		An open IPMP connection handle
cbMessage	dword		The count, in bytes, of the message pointed to by pbMessage
pbMessage	byte		The message content to send into e*Gate
cSecondsToExpire	dword	yes	The number of **seconds** this request “lives” within the e*Gate system before being dropped as an “expired” Event. If it is set to EWIPMP_NOEXPIRE, the message never expires.
dwflags	dword		Bit flags reserved for future use. This field ***must*** be set to zero.
pvReserved	void		Reserved for future use and ***must*** be set to null.

Return Values

Returns **true** if the message was successfully sent; otherwise, returns **false**. Use **getlasterror()** to obtain any error code.

Examples

```
// send the message to multiplexer e*way
    //

    dwMessageLength = dwCurrentBuffLen;

    if (!EWIPMP_Send(hIPMP, dwMessageLength, (BYTE*)pBuffer,
                     cSecondsToExpire, 0, NULL))
    {
        goto SendFailed;
```

EWIPMP_Wait

Syntax

```
EWIPMP_Wait(hIPMP,pcbReturnMessage,ppbReturnMessage,
             cMillisecondToExpire,dwflags,pvReserved)
```

Description

EWIPMP_Wait causes the application to wait the specified number of milliseconds for a response to be received via the specified message handle. A message must have been sent on the same HEWIPMP handle for which EWIPMP_Wait is invoked.

Parameters

Name	Type	Description
hIPMP	HEWIPMP	An IPMP connection handle.
pcbReturnMessage	dword	A pointer to a DWORD that receives the count, in bytes, of the returned message pointed to by ppbReturnMessage.
ppbReturnMessage	dword	The address of a **byte** pointer that is allocated and filled out by this API of the message content received. The caller must free the returned pointer using EWIPMP_Free (see **EWIPMP_Free** on page 252).
cMillisecondToExpire	dword	The number of **milliseconds** to wait to receive a message from the remote host. If the caller sets this parameter to EWIPMP_BLOCKWAIT, this API is not returned until a message is received or the connection is closed.
dwflags	dword	Bit flags reserved for future use. This ***must*** be set to zero.
pvReserved	void	Reserved for future use. This ***must*** be set to null.

Return Values

Returns **true** if the message was received properly; returns **false** if an error occurred or if the timeout expired. Use **getlasterror()** to obtain any error codes.

If the timeout expires, the error code is set to GENERROR_TIMEOUT. Other uncommon error codes that it might return from GETLASTERROR are:

GENERROR_TIMEOUT	0x20000040
GENERROR_INVALID_PARAM	0x20000002
GENERROR_BADFORMAT	0x20000050
GENERROR_MEMORY_ALLOCATION	0x20001000

Examples

```
// wait for reply
    //

    if (!EWIPMP_Wait(hIPMP, &dwMessageLength, (BYTE**)&pBuffer,
                     cMillisecondsToWait, 0, NULL))
    {
        goto WaitFailed;
    }
```

8.2 COBOL APIs

Open

Syntax

```
call "MUXAPI" using
     MUXAPI-handle
     MUXAPI-remote-host
     MUXAPI-remote-port
     MUXAPI-errno
     MUXAPI-retcode.
```

Description

This function creates a socket connection to the MUX server e*Way running on the specified remote host and TCP/IP port. This socket connection is defined by a unique identifier, or "handle," that is returned by the OPEN. Note this allows multiple connections to be opened and maintained by a single CICS application to a single or multiple MUX server e*Ways.

Sample

Sample working storage definitions:

```
01    MUXAPI-handle       pic s9(8)   binary value +0.
01   MUXAPI-remote-host   pic  x(24)  value 'remote.host.name'.
01   MUXAPI-remote-port   pic  9(8)   binary value 26051.
01   MUXAPI-errno         pic  9(8)   binary value 0.
01   MUXAPI-retcode       pic s9(8)   binary value +0.
```

Parameters set by the application:

MUXAPI - handle

A 4-byte binary number, initialized to zero.

Returns

TCP/IP socket number for the established connection.

MUXAPI - remote-host

A 24-byte character field, containing the DNS name of the remote host on which the MUX server e*Way is listening.

Returns

Unchanged.

MUXAPI - remote-port

A 4-byte binary number, containing the TCP/IP port number to which the MUX server e*Way is listening.

Returns

Unchanged

MUXAPI - errno

A 4-byte binary number, initialized to zero.

Returns

If MUXAPI - retcode is negative, this contains an error number.

MUXAPI - retcode

A 4-byte signed binary number, initialized to zero.

Returns

Negative value signifies an error.

Send

Syntax

```
call "MUXAPIS" using
     MUXAPI-handle
     MUXAPI-message-len
     MUXAPI-message
     MUXAPI-hsecs-for-ack
     MUXAPI-errno
     MUXAPI-retcode.
```

Description

This function sends a message or block of data to the MUX server e*Way. The function will then wait a specified time (expressed in hundredths of seconds) for an acknowledgment to arrive on the socket connection identified by the passed handle.

Sample

Sample working storage definitions:

```
01   MUXAPI-handle          pic s9(8)      binary.
01   MUXAPI-message-len     pic  9(8)      binary.
01   MUXAPI-message         pic  x(32703)  value spaces.
01   MUXAPI-hsecs-for-ack   pic  9(8)      binary.
01   MUXAPI-errno           pic  9(8)      binary value 0.
01   MUXAPI-retcode         pic s9(8)      binary value +0.
```

Parameters set by the application:

MUXAPI - handle

A 4-byte binary number containing the socket number returned by the OPEN.

Returns

Unchanged.

MUXAPI - message-len

A 4-byte binary number containing the length, in bytes, of the message to be sent to the MUX server e*Way. The maximum size is 32K - 40 bytes, or 32727 bytes.

Returns

Unchanged.

MUXAPI - message

A 32727-byte character field containing the actual data to be sent to the MUX server e*Way. The contents of this field will be transmitted without conversion of any kind.

Returns

Unchanged.

MUXAPI - hsecs-for-ack

A 4-byte binary number, initialized to zero. Hundredths of seconds to wait for an acknowledgement (ACK) from e*Gate after a SEND.

Returns

Unchanged.

MUXAPI - errno

A 4-byte binary number, initialized to zero.

Returns

If MUXAPI - retcode is negative, this contains an error number.

MUXAPI - retcode

A 4-byte signed binary number, initialized to zero.

Returns

Negative value signifies an error.

Receive

Syntax

```
call "MUXAPIR" using
     MUXAPI-handle
     MUXAPI-returnmsg-len
     MUXAPI-returnmsg
     MUXAPI-hsecs-to-wait
     MUXAPI-errno
     MUXAPI-retcode.
```

Description

This function receives a message or block of data from the MUX server e*Way. The function will wait a specified time (expressed in hundredths of seconds) for a message to arrive on the socket connection identified by the passed handle.

Sample

Sample working storage definitions:

```
01   MUXAPI-handle          pic s9(8)      binary.
01   MUXAPI-returnmsg-len   pic  9(8)      binary.
01   MUXAPI-returnmsg       pic  x(32727)  value spaces.
01   MUXAPI-hsecs-to-wait   pic  9(8)      binary value 100.
01   MUXAPI-errno           pic  9(8)      binary value 0.
01   MUXAPI-retcode         pic s9(8)      binary value +0.
```

Parameters set by the application:

MUXAPI - handle

A 4-byte binary number, containing the socket number returned by the OPEN.

Returns

Unchanged.

MUXAPI - returnmsg-len

A 4-byte binary number, initialized to zero.

Returns

The length, in bytes, of the data received from the MUX server e*Way.

MUX API - returnmsg

A 32727-byte character field.

Returns

The data received from the MUX server e*Way.

MUXAPI - hsecs-to-wait

A 4-byte binary number, representing the hundredths of seconds to wait for a response from e*Gate.

Returns

Unchanged.

MUXAPI - errno

A 4-byte binary number, initialized to zero.

Returns

If MUXAPI - retcode is negative, this contains an error number.

MUXAPI - retcode

A 4-byte signed binary number, initialized to zero.

Returns

Negative value signifies an error.

Close

Syntax

```
call "MUXAPIC" using
     MUXAPI-handle
     MUXAPI-errno
     MUXAPI-retcode.
```

Description

The close function shuts down the socket connection with the MUX server e*Way and frees any resources associated with it.

Sample

Sample working storage definitions:

```
01   MUXAPI-handle    pic s9(8)   binary.
01   MUXAPI-errno     pic  9(8)   binary value 0.
01   MUXAPI-retcode   pic s9(8)   binary value +0.
```

MUXAPI - handle

A 4-byte binary number, containing the socket number returned by the OPEN.

Returns

Unchanged.

MUXAPI - errno

A 4-byte binary number, initialized to zero.

Returns

If MUXAPI - retcode is negative, this contains an error number.

MUXAPI - retcode

A 4-byte signed binary number, initialized to zero.

Returns

Negative value signifies an error.

8.3 ActiveX APIs

The e*Gate API Kit ActiveX control supports the following methods:

- **Connect** on page 261
- **Disconnect** on page 261
- **LastErrorCode** on page 262
- **LastErrorText** on page 262
- **ReplyMessageAsArray** on page 263

- **ReplyMessageAsString** on page 263
- **ReplyMessageSize** on page 263
- **Send** on page 264
- **Wait** on page 264

Connect

Syntax

```
Connect bstrMUXHost, lMUXListenPort
```

Description

Connect opens a connection to the specified host using the specified port.

Parameters

Name	Type	Description
bstrMUXHost	BSTR	The name of a network host.
lMUXListenPort	long	The TCP/IP port number over which to establish the connection.

Return Values

None.

Examples

```
rr.Connect strHost, dwPort
```

Disconnect

Syntax

```
Disonnect
```

Description

Disconnect closes an open connection.

Parameters

None.

Return Values

None.

Examples

```
rr.Disconnect()
```

LastErrorCode

Syntax

```
LastErrorCode
```

Description

LastErrorCode returns the last error code.

Parameters

None.

Return Values

Returns an error code.

Examples

```
rr.Send strSend, 1000

    if rr.LastErrorCode() > 0 then
    ShowError(rr)
    else
    rr.Wait 10000

    if rr.LastErrorCode() > 0 then
    ShowError(rr)
    else
    Response.Write "<H1 align=center>MUX e*Way Response string</H1>"
    Response.Write "<P>"
```

LastErrorText

Syntax

```
LastErrorText
```

Description

LastErrorText returns the text of the last error code.

Parameters

None.

Return Values

Returns an error message.

Examples

```
rr.Send strSend, 1000

    if rr.LastErrorCode() > 0 then
    ShowError(rr)
    else
    rr.Wait 10000

    if rr.LastErrorCode() > 0 then
    ShowError(rr)
```

```
        else
        Response.Write "<H1 align=center>MUX e*Way Response string</H1>"
        Response.Write "<P>"
```

ReplyMessageAsArray

Syntax

```
ReplyMessageAsArray
```

Description

ReplyMessageAsArray returns the outbound data as an array.

Parameters

None.

Return Values

Returns an array.

ReplyMessageAsString

Syntax

```
ReplyMessageAsString
```

Description

ReplyMessageAsString returns the outbound data as a string.

Parameters

None.

Return Values

Returns a string.

Examples

```
rr.Wait 10000

    if rr.LastErrorCode() > 0 then
    ShowError(rr)
    else
    Response.Write "<H1 align=center>MUX e*Way Response string</H1>"
     Response.Write "<P>"
     Response.Write(rr.ReplyMessageAsString)
     Response.Write "</P>"
```

ReplyMessageSize

Syntax

```
ReplyMessageSize
```

Description

ReplyMessageSize returns the length in bytes of the outbound data.

Parameters

None.

Return Values

Returns a long integer.

Send

Syntax

```
Send bstrRequestMessage, cSecondsAlive
```

Description

Send sends data into the e*Gate system.

Parameters

Name	Type	Description
bstruRequestMessage	BSTR	The message to send into the e*Gate system.
cSecondsAlive	long	The number of **seconds** this request "lives" within the e*Gate system before being dropped as an "expired" Event.

Return Values

None.

Examples

```
rr.Send strSend, 1000
```

Wait

Syntax

```
Wait cBlockMilliseconds
```

Description

Wait causes the application to wait the specified number of milliseconds for a message to be received.

Parameters

Name	Type	Description
cBlockMilliseconds	BSTR	The number of milliseconds to wait to receive a message from the remote host.

Return Values

None.

Examples

```
rr.Wait 10000
```

8.3.1 ActiveX Class ID

The ID for the IP Multiplexing ActiveX control is

xipmpclnt.MUX

8.4 Java Methods

The Java class file IPMPReqReply.java defines the following classes:

Defaults

The IPMPReqReply class establishes the following default variables and values:

Variable	Type	Value	Meaning
port	long	26051	The TCP/IP port over which the connection to the remote host is established.
host	String	"" (null string)	The name of the remote host.
timeOut	long	10000	The number of milliseconds the application waits to receive a message from the remote host.
connectionHandle	long	0	A connection handle to the remote host.
secondsToExpire	long	10	The number of seconds that the message may travel within the e*Gate system before e*Gate marks it as "expired."

connect

Syntax

```
connect()
```

Description

connect establishes an IPMP connection with a remote host.

Type

Boolean

Parameters

None.

Return Values

Boolean
Returns **true** if the connection was established properly; otherwise, returns **false**.

Examples

```
// attempt to connect
    result = mux.connect();
    if (result == false)
            System.out.println("Unable to connect");
```

disconnect

Syntax

```
disconnect()
```

Description

disconnect closes an IPMP connection.

Type

Boolean

Parameters

None.

Return Values

Boolean
Returns **true** if the connection was broken properly; otherwise, returns **false**.

Examples

```
// close our connection
        result = mux.disconnect();
        if (result == false)
            System.out.println("Unable to close");

    System.exit(0);
```

getHost

Syntax

```
getHost()
```

Description

getHost returns the name of the current host as defined by the class file's global variable **host**. If no host name is defined, **getHost** returns a null string.

Type

String

Parameters

None.

Return Values

java.lang.String
Returns a host name if one is defined; otherwise, returns a null string.

Examples

```
host = mux.getHost();
```

getPort

Syntax

```
getPort()
```

Description

getPort returns the TCP/IP port number defined by the class file's global variable **port**.

Type

long

Parameters

None.

Return Values

long
Returns a port number.

Examples

```
port = mux.getPort();
```

getResponse

Syntax

```
getResponse()
```

Description

getResponse polls the remote system and returns that system's response.

Type

String

Parameters

None.

Return Values

java.lang.String
Returns the remote system's response. If no response is received or there is no connection handle, returns a null string.

Examples

```
// retrieve our response
    message = mux.getResponse();
```

getResponseBytes

Syntax

```
getResponseBytes()
```

Description

getResponseBytes polls the remote system and returns that system's response. The response is returned as a blob (unlike **getResponse** on page 268, which packages the response as a string).

Type

byte

Parameters

None.

Return Values

byte array
Returns the remote system's response as a byte array. If no response is received or there is no connection handle, returns a null string.

Examples

```
byte[] returnBytest = mux.getResponseByte();
```

getSecondsToExpire

Syntax

```
getSecondsToExpire()
```

Description

getSecondsToExpire returns the expiration time (in seconds) as defined by the class file's global variable **secondsToExpire**. See **"Defaults" on page 266** for more information.

Type

long

Parameters

None.

Return Values

long
Returns the expiration time (in seconds).

Examples

```
secondsToExpire = mux.getSecondsToExpire();
```

getSleepDuration

Syntax

```
getSleepDuration()
```

Description

getSleepDuration obtains the current internal sleep interval for MUX reply waiting.

Type

long

Parameters

None.

Return Values

long
Returns the sleep interval(in milliseconds).

Examples

```
long sleepduration = mux.getSleepDuration();
```

setSleepDuration

Syntax

```
getSleepDuration(long sleep)
```

Description

getSleepDuration obtains the current internal sleep interval for MUX reply waiting.

Type

long

Parameters

Name	Type	Description
sleep	long	The time in milliseconds to set the duration.

Return Values

long
Returns the sleep interval(in milliseconds).

Examples

```
long sleepduration = mux.getSleepDuration();
```

getTimeout

Syntax

```
getTimeout()
```

Description

getTimeout returns the timeout period (in milliseconds) as defined by the class file's global variable **timeOut**. See **"Defaults" on page 266** for more information.

Type

long

Parameters

None.

Return Values

long
Returns the timeout period (in milliseconds).

Examples

```
timeOut = mux.getTimeout();
```

sendMessage

Syntax

```
sendMessage(byte[] message_bytes)
```

Description

sendMessage sends the specified message to the remote host.

Type

Boolean

Parameters

Name	Type	Description
message_bytes	byte[ ] or String	The message to send.

Return Values

Boolean
Returns **true** if a non-null message was sent successfully; otherwise, returns **false**.

Examples

```
// send our request
    message = new String("Hello");
    result = mux.sendMessage(message);
    if (result == false)
    System.out.println("Message was not sent successfully");
    else
    System.out.println("Message was sent successfully");
```

setDebug

Syntax

```
setDebug(boolean mode)
```

Description

setDebug controls the print capability for debugging messages to System.out. By default, it is not enabled.

Type

void

Parameters

Name	Type	Description
mode	boolean	true to enable debugging method, false to surpress.

Return Values

None.

Examples

```
mux.setDebug(true);
```

setHost

Syntax

```
setHost(host_name)
```

Description

setHost sets the name of the remote host to the specified value.

Type

void

Parameters

Name	Type	Description
host_name	String	The new host name.

Return Values

None.

Examples

```
mux.setHost("localhost");
```

setPort

Syntax

```
setPort(port_number)
```

Description

setPort sets the TCP/IP port number to the specified value.

Type

void

Parameters

Name	Type	Description
port_number	long	The new port number.

Return Values

None.

Examples

```
mux.setPort(26051);
```

setSecondsToExpire

Syntax

```
setSecondsToExpire(seconds)
```

Description

setSecondsToExpire sets the expiration time to the specified value. See **"Defaults" on page 266** for more information.

Type

void

Parameters

Name	Type	Description
seconds	long	The new expiration time.

Return Values

None.

Examples

```
mux.setSecondsToExpire(10);
```

setTimeout

Syntax

```
setTimeout(milliseconds)
```

Description

setTimeout sets the timeout to the specified value. See **"Defaults" on page 266** for more information.

Type

void

Parameters

Name	Type	Description
milliseconds	long	The new timeout.

Return Values

None.

Examples

```
mux.setTimeout(10000);  // this is in milliseconds
```

8.5 com.stc.MUXPooler

The `MUXPooler` class operates between the multi-plexing e*Way and the external application. The `MUXPooler` is opened with the configured number of connections regardless of the number of connected applications. These connections are maintained by the e*Way to improve performance (connection/disconnection overhead is removed). The applications connected to the `MUXPooler` share these connections. If all of the connections are occupied, when another application tries to connect to the `MUXPooler`, a "Waiting for a free MUX" or "No MUX Available" message is produced.

8.5.1 Constructors

There are three Constructors associated with the Muxpooler class that are used to instantiate an object:

- `public MUXPooler():`Instantiates the object only. Each of the additional attributes must be called individually.
  - **setHost** on page 280
  - **setPort** on page 280
  - **setTimeout** on page 281
  - **setSecondsToExpire** on page 281
- `public MUXPooler(String host, int port, int connectionCount,int timeout, int secondsToExpire):` Instantiates the object and sets the values of the specified attributes.
- `public MUXPooler(String host, int port, int connectionCount,int timeout, int secondsToExpire, boolean debug):`Instantiates the object and sets the values of the specified attributes. Included in these attributes is the ability to print debugging code to System.out. By default it is not enabled.

8.5.2 Methods

This class will create a user defined number of MUX (multiplexer) connections to e*Gate and send/receive Events to e*Gate:

connect on page 276	**resizeMUXPool** on page 278
disconnect on page 276	**sendBytes** on page 279
disconnect on page 276	**sendMessage** on page 279
getConnectionCount on page 277	**setConnectionCount** on page 280
getHost on page 277	**setHost** on page 280
getPort on page 277	**setPort** on page 280
getSecondsToExpire on page 278	**setSecondsToExpire** on page 281

connect

Syntax

```
connect()
```

Description

connect opens a connection to the Participating Host that is running the MUX e*Way.

Parameters

None.

Return Values

Boolean
Returns true if the command executed successfully; otherwise, returns false.

disconnect

Syntax

```
disconnect()
```

Description

disconnect closes the connection to the Participating Host that is running the MUX e*Way and waits for each connection to finish the associated transaction.

Parameters

None.

Return Values

Boolean
Returns true if the command executed successfully; otherwise, returns false.

disconnect

Syntax

disconnect(*WaitOnMux*)

Description

disconnect disconnects all connections to the MUX e*Way.

Parameters

Name	Type	Description
WaitOnMux	Boolean	Determines whether to wait for current transactions to complete before disconnecting.

Return Values

Boolean
Returns true if the command executed successfully; otherwise, returns false.

getConnectionCount

Syntax

```
getConnectionCount()
```

Description

getConnectionCount returns the number of MUX connections currently available.

Parameters

None.

Return Values

integer
Returns the total number of connections available within the **`MUXPooler`**. This includes free, non-used connections as well as the occupied connections.

getHost

Syntax

```
getHost()
```

Description

getHost returns the host name.

Parameters

None.

Return Values

string
Returns the host name.

getPort

Syntax

```
getPort()
```

Description

getPort returns the port number of the host machine.

Parameters

None.

Return Values

integer
Returns the port number.

getSecondsToExpire

Syntax

```
getSecondsToExpire()
```

Description

getSecondsToExpire returns the expiration time in seconds.

Parameters

None.

Return Values

integer
Returns the number of milli-seconds.

getTimeout

Syntax

```
getTimeout()
```

Description

getTimeout returns the number of milli-seconds to wait for a response before timeout.

Parameters

None.

Return Values

integer
Returns the number of milli-seconds.

resizeMUXPool

Syntax

```
resizeMUXPool(newSize)
```

Description

resizeMUXPool resizes the muxPool to the specified size.

Parameters

Name	Type	Description
newSize	integer	The number of connections of the muxPool

Return Values

Boolean

Returns true if the command executed successfully; otherwise, returns false.

Additional Information

resizeMUXPool is used to change the pool size at runtime (as necessary).

sendBytes

Syntax

```
sendBytes(bytes_array)
```

Description

sendBytes takes the message (Event) that is to be delivered into e*Gate, and returns e*Gate's response. A null string is returned if there is no response.

Parameters

Name	Type	Description
bytes_array	byte array	The message (Event) as a byte array

Return Values

byte array

Returns a byte array containing e*Gate's response if available; otherwise, returns null.

sendMessage

Syntax

```
sendMessage(message)
```

Description

sendMessage takes the message (Event) that is to be delivered into e*Gate, and returns e*Gate's response. A null string is returned if there is no response.

Parameters

Name	Type	Description
message	string	The message (Event) to send

Return Values

string

Returns string containing e*Gate's response; otherwise, returns null if there was no response.

setConnectionCount

Syntax

```
setConnectionCount(count)
```

Description

setConnectionCount sets the number of MUX connections created.

Parameters

Name	Type	Description
count	integer	The number of MUX connections

Return Values

void

Additional Information

setConnectionCount is used to initialize the Class.

setHost

Syntax

```
setHost(host)
```

Description

setHost specifies the host name with which to establish connection.

Parameters

Name	Type	Description
host	string	The host name.

Return Values

void

setPort

Syntax

```
setPort(integer)
```

Description

setPort specifies the port number with which to establish connection.

Parameters

Name	Type	Description
port	integer	The port name.

Return Values

void

setSecondsToExpire

Syntax

```
setSecondsToExpire(seconds)
```

Description

setSecondsToExpire sets the expiration time to the specified value.

Parameters

Name	Type	Description
seconds	integer	The number of seconds

Return Values

void

setTimeout

Syntax

```
setTimeout(timeout)
```

Description

setTimeout sets the timeout to the specified value.

Parameters

Name	Type	Description
timeout	integer	The number of seconds

Return Values

void

8.6 Perl Subroutines

The e*Gate API Kit supports the following Perl extension subroutines:

- **LastErrorText** on page 262
- **LastErrorCode** on page 262
- **Connect** on page 261
- **Send** on page 264
- **Multiplexer_Send** on page 284
- **ReplyMessageAsArray** on page 263
- **Wait** on page 264

Multiplexer_Close

Syntax

```
Multiplexer_Close(handle)
```

Description

Multiplexer_Close closes the connection on the specified handle.

Parameters

Name	Type	Description
handle	handle	The handle returned by Multiplexer_Open

Return Values

integer
Returns **1** if the command executed successfully; otherwise, returns **0**.

Example

```
$result = Multiplexer_Close($handle);
if(!$result)
{
    print "Multiplexer_Close failed.\n";
}
```

Multiplexer_Free

Syntax

```
Multiplexer_Free(handle, message-pointer)
```

Description

Multiplexer_Free frees the memory associated with the message pointer.

Parameters

Name	Type	Description
handle	handle	The handle returned by Multiplexer_Open
message-pointer	pointer	The message pointer

Return Values

integer
Returns **1** if the command executed successfully; otherwise, returns **0**.

Examples

```
$message_ptr = Multiplexer_Wait($handle,$message_length,3000, 0, 0);

$result = Multiplexer_Free($handle, $message_ptr);

if(!$result)
{
    print "Multiplexer_Free failed.\n";
}
```

Multiplexer_Init

Syntax

```
Multiplexer_Init(dll-path)
```

Description

Multiplexer_Init specifies the path that contains the e*Way's library files. This function is required in every Perl script that communicates with the Participating Host.

Parameters

Name	Type	Description
dll-path	string	The path that contains the e*Way's **.dll** files. The path can be a relative or absolute path to the script that calls the function.

Return Values

integer
Returns **1** if the command executed successfully; otherwise, returns **0**.

Examples

```
# this is where stc_ewipmpclntperl.dll is located.
use lib ("egate/client/bin");

use CGI qw/:standard/;
use stc_ewipmpclntperl.dll;

# call Multiplexer_Init to define the dll path
Multiplexer_Init("egate/client/bin");
```

Multiplexer_Open

Syntax

```
Multiplexer_Open(server-name, server-port, flags, reserved)
```

Description

Multiplexer_Open opens a connection to the Participating Host that is running the IPMP e*Way.

Parameters

Name	Type	Description
server-name	string	The name of the e*Gate Participating Host
server-port	integer	The port through which to communicate with the multi-plexing e*Way. If this value is zero, the default port **26051** is used.
flags	integer	Command flags. always set to “0” (zero) unless directed otherwise by SeeBeyond support personnel.
reserved	integer	Reserved for future SeeBeyond use. always set to “0” (zero) unless directed otherwise by SeeBeyond support personnel.

Return Values

Returns a connection handle.

Examples

```
$handle = Multiplexer_Open(“server.mycompany.com”,26051, 0, 0);
  if(!$handle)
  { print "Multiplexer_Open failed.\n"; }
```

Multiplexer_Send

Syntax

```
Multiplexer_Send(handle, message-length, message,seconds-to-expire,
flags, reserved)
```

Description

Multiplexer_Send sends the specified message to the e*Gate Participating Host.

Parameters

Name	Type	Description
handle	handle	The handle returned by Multiplexer_Open.
message-length	integer	The length of the message, in bytes.
message	string	The data to send to the e*Gate Participating Host.
seconds-to-expire	integer	The number of seconds after which the data within the e*Gate system expires.
flags	integer	Command flags. always set to “0” (zero) unless directed otherwise by SeeBeyond support personnel.
reserved	integer	Reserved for future SeeBeyond use. always set to “0” (zero) unless directed otherwise by SeeBeyond support personnel.

Return Values

integer
Returns **1** if the command executed successfully; otherwise, returns **0**.

Examples

```
$result = Multiplexer_Send($handle, $message_length,$message,0,0,0);
if(!$result)
{
    print "STC_MUX::Multiplexer_Send failed.\n";
}
```

Multiplexer_ToString

Syntax

Multiplexer_ToString(*message-pointer*)

Description

Multiplexer_ToString returns the data associated with the specified message pointer as a string.

Parameters

Name	Type	Description
message-pointer	pointer	The pointer returned by Multiplexer_Wait

Return Values

integer
Returns **1** if the command executed successfully; otherwise, returns **0**.

Additional Notes

In the current implementation, the null character ("\0") terminates a response message, and any information that follows a null character is discarded when you use **Multiplexer_ToString** to convert the response message to a string.

Examples

```
$message_received = Multiplexer_ToString($message_ptr);
$result = Multiplexer_Free($handle, $message_ptr);
if(!$result)
{
    print "Multiplexer_Free failed.\n";
}
```

See **LastErrorCode** on page 262 for more information.

Multiplexer_Wait

Syntax

```
Multiplexer_Wait(handle, message-length,millsecond-timeout, flags,
reserved)
```

Description

Multiplexer_Wait causes the application to wait up to the specified number of milliseconds for a message to be received.

Parameters

Name	Type	Description
handle	handle	The handle returned by Multiplexer_Open
message-length	integer	The length of the message received, in bytes (assigned as an output parameter).
millisecond-timeout	integer	The number of milliseconds to wait
flags	integer	Command flags. always set to "0" (zero) unless directed otherwise by SeeBeyond support personnel.
reserved	integer	Reserved for future SeeBeyond use. always set to "0" (zero) unless directed otherwise by SeeBeyond support personnel.

Return Values

Returns a message pointer.

Examples

```
$message_ptr = Multiplexer_Wait($handle,$message_length,3000, 0, 0);
if(!$message_ptr)
{
    print "Multiplexer_Wait failed.\n";
}
```

Appendix A

Appendix

A.1 Cobol API Return Codes

The following error codes have been created for the e*Gate API Kit's Cobol support.

Error Code	Description	Programmer Response
3001	Get Host Name Error.	Be sure that the host name and port number are correct.
3002	Error on Return from ezacic08.	Internal error. Call your system administrator.
3003	No Data Received - Socket Closed	Restart connection.
3004	Partial Header Received.	Retry. If problem persists, call your system administrator.
3005	Timed out waiting for a received.	Increase the timeout interval and retry. Note: Remember that the timeout value is in hundredths of a second. One second would be passed as a value of 100.

A.2 Cobol Error Return Codes

The following return codes can be found in the *IP CICS Sockets Guide Version 2 Release 8 and 9, OS/390 SecureWay Communications Server*. As of the date of the creation of this document, it can be downloaded from:

http://www-1.ibm.com/servers/s390/os390/bkserv/r10pdf/commserv.html

A.2.1 TCP/IP for MVS Return Codes

Error Number	Message Name	Socket Type	Error Description	Programmer's Response
1	EPERM	All	Permission is denied. No owner exists.	Check that TCP/IP is still active; Check the protocol value of the socket call.
1	EDOM	All	Argument is too large.	Check parameter values of the function call.
2	ENOENT	All	The data set or directory was not found.	Check files used by the function call.
2	ERANGE	All	The result is too large.	Check parameter values of the function call.
3	ESRCH	All	The process was not found. A table entry was not located.	Check parameter values and structures pointed to by the function parameters.
4	EINTR	All	A system call was interrupted.	Check that the socket connection and TCP/IP are still active.
5	EIO	All	An I/O error occurred.	Check status and contents of source database if this occurred during a file access.
6	ENXIO	All	The device or driver was not found.	Check status of the device attempting to access.
7	E2BIG	All	The argument list is too long.	Check the number of function parameters.
8	ENOEXEC	All	An EXEC format error occurred.	Check that the target module on an exec call is a valid executable module.
9	EBADF	All	An incorrect socket descriptor was specified.	Check socket descriptor value. It might be currently not in use or incorrect.
9	EBADF	Givesocket	The socket has already been given. The socket domain is not AF_INET.	Check the validity of function parameters.
9	EBADF	Select	One of the specified descriptor sets is an incorrect socket descriptor.	Check the validity of function parameters.
9	EBADF	Takesocket	The socket has already been taken.	Check the validity of function parameters.
10	ECHILD	All	There are no children.	Check if created subtasks still exist.
11	EAGAIN	All	There are no more processes.	Retry the operation. Data or condition might not be available at this time.
12	ENOMEM	All	There is not enough storage.	Check validity of function parameters.

Error Number	Message Name	Socket Type	Error Description	Programmer's Response
13	EACCES	All	Permission denied, caller not authorized.	Check access authority of file.
13	EACCES	Takesocket	The other application (listener) did not give the socket to your application. Permission denied, caller not authorized.	Check access authority of file.
14	EFAULT	All	An incorrect storage address or length was specified.	Check validity of function parameters.
15	ENOTBLK	All	A block device is required.	Check device status and characteristics.
16	EBUSY	All	Listen has already been called for this socket. Device or file to be accessed is busy.	Check if the device or file is in use.
17	EEXIST	All	The data set exists.	Remove or rename existing file.
18	EXDEV	All	This is a cross-device link. A link to a file on another file system was attempted.	Check file permissions.
19	ENODEV	All	The specified device does not exist.	Check file name and if it exists.
20	ENOTDIR	All	The specified device does not exist.	Use a valid file that is a directory.
21	EISDIR	All	The specified directory is a directory.	Use a valid file that is not a directory.
22	EINVAL	All types	An incorrect argument was specified.	Check validity of function parameters.
23	ENFILE	All	Data set table overflow occurred.	Reduce the number of open files.
24	EMFILE	All	The socket descriptor table is full.	Check the maximum sockets specified in MAXDESC().
25	ENOTTY	All	An incorrect device call was specified.	Check specified IOCTL() values.
26	ETXTBSY	All	A text data set is busy.	Check the current use of the file.
27	EFBIG	All	The specified data set is too large.	Check size of accessed dataset.
28	ENOSPC	All	There is no space left on the device.	Increase the size of accessed file.
29	ESPIPE	All	An incorrect seek was attempted.	Check the offset parameter for seek operation.
30	EROFS	All	The data set system is Read only.	Access data set for read only operation.
31	EMLINK	All	There are too many links.	Reduce the number of links to the accessed file.
32	EPIPE	All	The connection is broken. For socket write/send, peer has shutdown one or both directions.	Reconnect with the peer.

Error Number	Message Name	Socket Type	Error Description	Programmer's Response
33	EDOM	All	The specified argument is too large.	Check and correct function parameters.
34	ERANGE	All	The result is too large.	Check parameter values.
35	EWOULDBLOCK	Accept	The socket is in nonblocking mode and connections are not queued. This is not an error condition.	Reissue Accept( ).
35	EWOULDBLOCK	Read Recvfrom	The socket is in nonblocking mode and read data is not available. This is not an error condition.	Issue a select on the socket to determine when data is available to be read or reissue the Read( )/Recvfrom( ).
35	EWOULDBLOCK	Send Sendto Write	The socket is in nonblocking mode and buffers are not available.	Issue a select on the socket to determine when data is available to be written or reissue the Send( ), Sendto( ), or Write( ).
36	EINPROGRESS	Connect	The socket is marked nonblocking and the connection cannot be completed immediately. This is not an error condition.	See the Connect( ) description for possible responses.
37	EALREADY	Connect	The socket is marked nonblocking and the previous connection has not been completed.	Reissue Connect( ).
37	EALREADY	Maxdesc	A socket has already been created calling Maxdesc( ) or multiple calls to Maxdesc( ).	Issue Getablesize( ) to query it.
37	EALREADY	Setibmopt	A connection already exists to a TCP/IP image. A call to SETIBMOPT (IBMTCP_IMAGE), has already been made.	Only call Setibmopt( ) once.
38	ENOTSOCK	All	A socket operation was requested on a nonsocket connection. The value for socket descriptor was not valid.	Correct the socket descriptor value and reissue the function call.
39	EDESTADDRREQ	All	A destination address is required.	Fill in the destination field in the correct parameter and reissue the function call.
40	EMSGSIZE	Sendto Sendmsg Send Write	The message is too long. It exceeds the IP limit of 64K or the limit set by the setsockopt( ) call.	Either correct the length parameter, or send the message in smaller pieces.
41	EPROTOTYPE	All	The specified protocol type is incorrect for this socket.	Correct the protocol type parameter.

Error Number	Message Name	Socket Type	Error Description	Programmer's Response
42	ENOPROTOOPT	Getsockopt Setsockopt	The socket option specified is incorrect or the level is not SOL_SOCKET. Either the level or the specified optname is not supported.	Correct the level or optname.
42	ENOPROTOOPT	Getibmsocket opt Setibmsocket opt	Either the level or the specified optname is not supported.	Correct the level or optname.
43	EPROTONOSUPPORT	Socket	The specified protocol is not supported.	Correct the protocol parameter.
44	ESOCKTNOSUPPORT	All	The specified socket type is not supported.	Correct the socket type parameter.
45	EOPNOTSUPP	Accept Givesocket	The selected socket is not a stream socket.	Use a valid socket.
45	EOPNOTSUPP	Listen	The socket does not support the Listen call.	Change the type on the Socket( ) call when the socket was created. Listen( ) only supports a socket type of SOCK_STREAM.
45	EOPNOTSUPP	Getibmopt Setibmopt	The socket does not support this function call. This command is not supported for this function.	Correct the command parameter. See Getibmopt( ) for valid commands. Correct by ensuing a Listen( ) was not issued before the Connect( ).
46	EPFNOSUPPORT	All	The specified protocol family is not supported or the specified domain for the client identifier is not AF_INET=2.	Correct the protocol family.
47	EAFNOSUPPORT	Bind Connect Socket	The specified address family is not supported by this protocol family.	For Socket( ), set the domain parameter to AF_INET. For Bind( ), and Connect( ), set Sin_Family in the socket address structure to AF_INET.
47	EAFNOSUPPORT	Getclient Givesocket	The socket specified by the socket descriptor parameter was not created in the AF_INET domain.	The Socket( ) call used to create the socket should be changed to use AF_INET for the domain parameter.
48	EADDRINUSE	Bind	The address is in a timed wait because a LINGER delay from a previous close or another process is using the address.	If you want to reuse the same address, use Setsocketopt( ) with SO_REUSEADDR. See Setsockopt( ). Otherwise, use a different address or port in the socket address structure.

Error Number	Message Name	Socket Type	Error Description	Programmer's Response
49	EADDRNOTAVAIL	Bind	The specified address is incorrect for this host.	Correct the function address parameter.
49	EADDRNOTAVAIL	Connect	The calling host cannot reach the specified destination.	Correct the function address parameter.
50	ENETDOWN	All	The network is down.	Retry when the connection path is up.
51	ENETUNREACH	Connect	The network cannot be reached.	Ensure that the target application is active.
52	ENETRESET	All	The network dropped a connection on a reset.	Reestablish the connection between the applications.
53	ECONNABORTED	All	The software caused a connection abend.	Reestablish the connection between the applications.
54	ECONNRESET	All	The connection to the destination host is not available.	
54	ECONNRESET	Send Write	The connection to the destination host is not available.	The socket is closing. Issue Send( ) or Write( ) before closing the socket.
55	ENOBUFS	All	No buffer space is available.	Check the application for massive storage allocation call.
55	ENOBUFS	Accept	Not enough buffer space is available to create the new socket.	Call your system administrator.
55	ENOBUFS	Send Sendto Write	Not enough buffer space is available to send the new message.	Call your system administrator.
56	EISCONN	Connect	The socket is already connected.	Correct the socket descriptor on Connect( ) or do not issue a Connect( ) twice for the socket.
57	ENOTCONN	All	The socket is not connected.	Connect the socket before communicating.
58	ESHUTDOWN	All	A Send cannot be processed after socket shutdown.	Issue read/receive before shutting down the read side of the socket.
59	ETOOMANYREFS	All	There are too many references. A splice cannot be completed.	Call your system administrator.
60	ETIMEDOUT	Connect	The connection timed out before it was completed.	Ensure the server application is available.
61	ECONNREFUSED	Connect	The requested connection was refused.	Ensure server application is available and at specified port.
62	ELOOP	All	There are too many symbolic loop levels.	Reduce symbolic links to specified file.

Error Number	Message Name	Socket Type	Error Description	Programmer's Response
63	ENAMETOOLONG	All	The file name is too long.	Reduce size of specified file name.
64	EHOSTDOWN	All	The host is down.	Restart specified host.
65	EHOSTUNREACH	All	There is no route to the host.	Set up network path to specified host and verify that host name is valid.
66	ENOTEMPTY	All	The directory is not empty.	Clear out specified directory and reissue call.
67	EPROCLIM	All	There are too many processes in the system.	Decrease the number of processes or increase the process limit.
68	EUSERS	All	There are to many users on the system.	Decrease the number of users or increase the user limit.
69	EDQUOT	All	The disk quota has been exceeded.	Call your system administrator.
70	ESTALE	All	An old NFS** data set handle was found.	Call your system administrator.
71	EREMOTE	All	There are too many levels of remote in the path.	Call your system administrator.
72	ENOSTR	All	The device is not a stream device.	Call your system administrator.
73	ETIME	All	The timer has expired.	Increase timer values or reissue function.
74	ENOSR	All	There are no more stream resources.	Call your system administrator.
75	ENOMSG	All	There is no message of the desired type.	Call your system administrator.
76	EBADMSG	All	The system cannot read the file message.	Verify that CS for OS/390 installation was successful and that message files were properly loaded.
77	EIDRM	All	The identifier has been removed.	Call your system administrator.
78	EDEADLK	All	A deadlock condition has occurred.	Call your system administrator.
78	EDEADLK	Select Selectex	None of the sockets in the socket descriptor sets is either AF_NET or AF_IUCV sockets, and there is no timeout or no ECB specified. The select/selectex would never complete.	Correct the socket descriptor sets so that an AF_NET or AF_IUCV socket is specified. A timeout of ECB value can also be added to avoid the select/selectex from waiting indefinitely.
79	ENOLCK	All	No record locks are available.	Call your system administrator.
80	ENONET	All	The requested machine is not on the network.	Call your system administrator.

Error Number	Message Name	Socket Type	Error Description	Programmer's Response
81	ERREMOTE	All	The object is remote.	Call your system administrator.
82	ENOLINK	all	The link has been severed.	Release the sockets and re initialize the client-server connection.
83	EADV	All	An ADVERTISE error has occurred.	Call your system administrator.
84	ESRMNT	All	AnSRMOUNT error has occurred.	Call your system administrator.
85	ECOMM	All	A communication error has occurred on a Send call.	Call your system administrator.
86	EPROTO	All	A protocol error has occurred.	Call your system administrator.
87	EMULTIHOP	All	A multi hop address link was attempted.	Call your system administrator.
88	EDOTDOT	All	A cross-mount point was detected. This is not an error.	Call your system administrator.
89	EREMCHG	all	The remote address has changed.	Call your system administrator.
90	ECONNCLOSED	All	The connection was closed by a peer.	Check that the peer is running.
113	EBADF	All	Socket descriptor is not in correct range. The maximum number of socket descriptors is set by MAXDESC( ). The default range is 0 - 49.	Reissue function with corrected socket descriptor.
113	EBADF	Bind socket	The socket descriptor is already being used.	Correct the socket descriptor.
113	EBADF	Givesocket	The socket has already been given. The socket domain is not AF_INET.	Correct the socket descriptor.
113	EBADF	Select	One of the specified descriptor sets is an incorrect socket descriptor.	Correct the socket descriptor. Set on Select( ) or Selectex( ).
113	EBADF	Takesocket	The socket has already been taken.	Correct the socket descriptor.
113	EBADF	Accept	A Listen( ) has not been issued before the Accept( )	Issue Listen( ) before Accept( ).
121	EINVAL	All	An incorrect argument was specified.	Check and correct all function parameters.
145	E2BIG	All	The argument list is too long.	Eliminate excessive number of arguments.
156	EMVSINITIAL	All	Process initialization error.	Attempt to initialize again.
1002	EIBMSOCKOUTOFRANGE	Socket	A socket number assigned by the client interface code is out of range.	check the socket descriptor parameter.
1003	EIBMSOCKINUSE	Socket	A socket number assigned by the client interface code is already in use.	Use a different socket descriptor.

Error Number	Message Name	Socket Type	Error Description	Programmer's Response
1004	EIBMIUCVERR	All	The request failed because of an IUCV error. This error is generated by the client stub code.	Ensure IUCV/VMCF is functional.
1008	EIBMCONFLICT	All	This request conflicts with a request already queued on the same socket.	Cancel the existing call or wait for its completion before reissuing this call.
1009	EIMBCANCELLED	All	The request was cancelled by the CANCEL call.	Informational, no action needed.
1011	EIBMBADTCPNAME	All	A TCP/IP name that is not valid was detected.	Correct the name specified in the IBM_TCPIMAGE structure.
1011	EIBMBADTCPNAME	Setibmopt	A TCP/IP name that is not valid was detected.	Correct the name specified in the IBM_TCPIMAGE.
1011	EIBMBADTCPNAME	INITAPI	A TCP/IP name that is not valid was detected.	Correct the name specification the IDENT option TCPNAME field.
1012	EIBMBADREQUESTCODE	All	A request code that is not valid was detected.	Contact your system administrator.
1013	EIBMBADCONNECTIONSTATE	All	A connection token that is not valid was detected; bad state.	Verify TCP/IP is active.
1014	EIBMUNAUTHORIZED CALLER	All	An unauthorized caller specified an authorized keyword.	Ensure user ID has authority for the specified operation.
1015	EIBMBADCONNECTIONMATCH	All	A connection token that is not valid was detected. There is no such connection.	Verify TCP/IP is active.
1016	EIBMTCPABEND	All	An abend occurred when TCP/IP was processing this request.	Verify that TCP/IP has restarted.
1026	EIBMINVDELETE	All	Delete requestor did not create the connection.	Delete the request from the process that created it.
1027	EIBMINVSOCKET	All	A connection token that is not valid was detected. No such socket exists.	Call your system programmer.
1028	EIBMINVTCPCONNECTION	All	Connection terminated by TCP/IP. The token was invalidated by TCP/IP.	Reestablish the connection to TCP/IP.
1032	EIBMCALLINPROGRESS	All	Another call was already in progress.	Reissue after previous call has completed.
1036	EIBMNOACTIVETCP	Getibmopt	No TCP/IP image was found.	Ensure TCP/IP is active.
1037	EIBMINVTSRBUSERDATA	All	The request control block contained data that is not valid.	check your function parameters and call your system programmer.
1038	EIBMINVUSERDATA	All	The request control block contained user data that is not valid.	Check your function parameters and call your system programmer.

Error Number	Message Name	Socket Type	Error Description	Programmer's Response
1040	EIBMSELECTEXPOST	SELECTEX	SELECTEX passed an ECB that was already posted.	Check whether the user's ECB was already posted.
2001	EINVALIDRXSOCKETCALL	REXX	A syntax error occurred in the RXSOCKET parameter list.	Correct the parameter list passed to the REXX socket call.
2002	ECONSOLEINTERRUPT	REXX	A console interrupt occurred.	Retry the task.
2003	ESUBTASKINVALID	REXX	The subtask ID is incorrect.	Correct the subtask ID on the INITIALIZE call.
2004	ESUBTASKALREADYACTIVE	REXX	The subtask is already active.	Only issue the INITIALIZE call once in your program.
2005	ESUBTASKALNOTACTIVE	REXX	The subtask is not active.	Issue the INITALIZE call before any other socket call.
2006	ESOCKETNOTALLOCATED	REXX	The specified socket could not be allocated.	Increase the user storage allocation for this job.
2007	EMAXSOCKETSREACHED	REXX	The maximum number of sockets has been reached.	Increase the number of allocate sockets, or decrease the number of sockets used by your program.
2009	ESOCKETNOTDEFINED	REXX	The socket is not defined.	Issue the SOCKET call before the call that fails.
2011	EDOMAINSERVERFAILURE	REXX	A Domain Name Server failure occurred.	Call your MVS system programmer.
2012	EINVALIDNAME	REXX	An incorrect name was received from the TCP/IP server.	Call your MVS system programmer.
2013	EINVALIDCLIENTID	REXX	An incorrect clientid was received from the TCP/IP server.	Call your MVS server.
2014	EINVALIDFILENAME	REXX	An error occurred during NUCEXT processing.	Specify the correct translation table file name, or verify that the translation table is valid.
2016	EHOSTNOTFOUND	REXX	The host is not found.	Call your MVS system programmer.
2017	EIPADDRNOTFOUND	REXX	Address not found.	Call your MVS system programmer.

A.2.2 Sockets Extended Return Codes

Error Code	Problem Description	System Action	Programmer's Response
10100	An ESTATE macro did not complete normally.	End the call.	Call your MVS system programmer.
10101	A STORAGE OBTAIN failed.	End the call.	Increase MVS storage in the application's address space.
10108	The first call from TCP/IP was not INITAPI or TAKESOCKET.	End the call.	Change the first TCP/IP call to INITAPI or TAKESOCKET.
10110	LOAD of EZBSOH03 (alias EZASOH03) failed.	End the call.	Call the IBM Software Support Center.
10154	Errors were found in the parameter list for an IOCTL call.	Disable the subtask for interrupts. Return an error code to the caller.	Correct the IOCTL call. You might have incorrect sequencing of socket calls.
10155	The length parameter for an IOCTL call is 3200 (32 x 100).	Disable the subtask for interrupts. Return an error code to the caller.	Correct the IOCTL call. You might have incorrect sequencing of socket calls.
10159	A zero or negative data length was specified for a READ or READV call.	Disable the subtask for interrupts. Return an error code to the caller.	Correct the length in the READ call.
10161	The REQARG parameter in the IOCTL parameter list is zero.	End the call.	Correct the program.
10163	A 0 or negative data length was found for a RECV, RECVFROM, or RECVMSG call.	Disable the subtask for interrupts. Sever the DLC path. Return an error code to the caller.	Correct the data length.
10167	The descriptor set size for SELECT or SELECTEX call is less than or equal to zero.	Disable the subtask for interrupts. Return an error code to the caller.	Correct the SELECT or SELECTEX call. You might have incorrect sequencing of socket calls.
10168	The descriptor set size in bytes for a SELECT or SELECTEX call is greater than 252. A number greater than the maximum number of allowed sockets (2000 is maximum) has been specified.	Disable the subtask for interrupts. Return an error code to the caller.	Correct the descriptor set size.
10170	A zero or negative data length was found for a SEND or SENDMSG call.	Disable the subtask for interrupts. Return an error code to the caller.	Correct the data length in the SEND call.
10174	A zero or negative data length was found for a SENDTO call.	Disable the subtask for interrupts. Return an error code to the caller.	correct the data length in the SENDTO call.
10178	The SETSOCKOPT option length is less than the minimum length.	Disable the subtask for interrupts. Return an error code to the caller.	Correct the OPTLEN parameter.
10179	The SETSOCKOPT option length is greater than the maximum length.	Disable the subtask for interrupts. Return an error code to the caller.	Correct the OPTLEN parameter.

Error Code	Problem Description	System Action	Programmer's Response
10184	A data length of zero was specified for a WRITE call.	Disable the subtask for interrupts. Return an error code to the caller.	Correct the data length in the WRITE call.
10186	A negative data length was specified for a WRITE or WRITEV call.	Disable the subtask for interrupts. Return an error code to the caller.	Correct the data length in the WRITE call.
10190	The GETHOSTNAME option length is less than 24 or greater than the maximum length.	Disable the subtask for interrupts. Return an error code to the caller.	Correct the length parameter.
10193	The GETSOCKOPT option length is less than the minimum or greater than the maximum length.	End the call.	Correct the length parameter.
10197	The application issued an INITAPI call after the connection was already established.	Bypass the call.	Correct the logic that produces the INITAPI call that is not valid.
10198	The maximum number of sockets specified for an INITAPI exceeds 2000.	Return to the user.	Correct the INITAPI call.
10200	The first call issued was not a valid first call.	End the call.	For a list of valid first calls, refer to the section on special considerations in the chapter on general programming.
10202	The RETARG parameter in the IOCTL call is zero.	End the call.	Correct the parameter list. You might have incorrect sequencing of socket calls.
10203	The requested socket number is a negative value.	End the call.	Correct the requested socket number.
10205	The requested socket number is a negative value.	End the call.	Correct the requested socket number.
10208	the NAMELEN parameter for a GETHOSTYNAME call was not specified.	End the call.	Correct the NAMELEN parameter. You might have incorrect sequencing of socket calls.
10209	The NAME parameter on a GETHOSTBYNAME call was not specified.	End the call.	Correct the NAME parameter. You might have incorrect sequencing of socket calls.
10210	The HOSTENT parameter on a GETHOSTBYNAME call was not specified.	End the call.	Correct the HOSTENT parameter. You might have incorrect sequencing of socket calls.
10211	The HOSTADDR parameter on a GETHOSTBYNAME or GETHOSTBYADDR call is incorrect.	End the call.	Correct the HOSTENT parameter. You might have incorrect sequencing of socket calls.
10212	The resolver program failed to load correctly for GETHOSTBYNAME or GETHOSTBYADDR call.	End the call.	Check the JOBLIB, STEPLIB, and link lib datasets and rerun the program.

Error Code	Problem Description	System Action	Programmer's Response
10213	Not enough storage is available to allocate the HOSTENT structure.	End the call.	Increase the use storage allocation for this job.
10214	The HOSTENT structure was not returned by the resolver program.	End the call.	Ensure that the domain name server is available. This can be a non error condition indicating that the name or address specified in a GETHOSTBYADDR or GETHOSTBYNAME call could not be matched.
10215	The APITYPE parameter on an INITAPI call instruction was not 2 or 3.	End the call.	Correct the APITYPE parameter.
10218	The application programming interface (API) cannot locate the specified TCP/IP.	End the call.	Ensure that an API that supports the performance improvements related to CPU conservation is installed on the system and verify that a valid TCP/IP name was specified on the INITAPI call. This error call might also mean that EZASOKIN could not be loaded.
10219	The NS parameter is greater than the maximum socket for this connection.	End the call.	Correct the NS parameter on the ACCEPT, SOCKET or TAKESOCKET call.
10221	The AF parameter of a SOCKET call is not AF_INET.	End the call.	Set the AF parameter equal of AF_INET.
10222	the SOCTYPE parameter of a SOCKET call must be stream, datagram, or raw (1, 2, or 3).	End the call.	Correct the SOCTYPE parameter.
10223	No ASYNC parameter specified for INITAPI with APITYPE=3 call.	End the call.	Add the ASYNC parameter to the INITAPI call.
10224	The IOVCNT parameter is less than or equal to zero, for a READV, RECVMSG, SENDMSG, or WRITEV call.	End the call.	correct the IOVCNT parameter.
10225	The IOVCNT parameter is greater than 120, for a READV, RECVMSG, SENDMSG, or WRITEV call.	End the call.	Correct the IOVCNT parameter.
10226	Invalid COMMAND parameter specified for a GETIBMOPT call.	End the call.	Correct the IOVCNT parameter.
10229	A call was issued on an APITYPE=3 connection without an ECB or REQAREA parameter.	End the call.	Add an ECB or REQAREA parameter to the call.
10300	Termination is in progress for either the CICS transaction or the sockets interface.	End the call.	None.

Error Code	Problem Description	System Action	Programmer's Response
10331	A call that is not valid was issued while in SRB mode.	End the call.	Get out of SRB mode and reissue the call.
10332	A SELECT call is invoked with a MAXSOC value greater than that which was returned in the INITAPI function (MAXSNO field).	End the call.	Correct the MAXSOC parameter and reissue the call.
10999	An abend has occurred in the subtask.	Write message EZY1282E to the system console. End the subtask and post the TRUE ECB.	If the call is correct, call your system programmer.
20000	An unknown function code was found in the call.	End the call.	Correct the SOC-FUNCTION parameter.
20001	The call passed an incorrect number of parameters.	End the call.	Correct the parameter list.
20002	The CICS Sockets Interface is not in operation.	End the call.	Start the CICS Sockets Interface before executing this call.

Index