

e*Gate API Kit Developer's Guide

*Release 5.0.5 for Schema Run-time
Environment (SRE)*



Copyright © 2005, 2010, Oracle and/or its affiliates. All rights reserved.

This software and related documentation are provided under a license agreement containing restrictions on use and disclosure and are protected by intellectual property laws. Except as expressly permitted in your license agreement or allowed by law, you may not use, copy, reproduce, translate, broadcast, modify, license, transmit, distribute, exhibit, perform, publish, or display any part, in any form, or by any means. Reverse engineering, disassembly, or decompilation of this software, unless required by law for interoperability, is prohibited.

The information contained herein is subject to change without notice and is not warranted to be error-free. If you find any errors, please report them to us in writing.

If this is software or related software documentation that is delivered to the U.S. Government or anyone licensing it on behalf of the U.S. Government, the following notice is applicable:

U.S. GOVERNMENT RIGHTS Programs, software, databases, and related documentation and technical data delivered to U.S. Government customers are "commercial computer software" or "commercial technical data" pursuant to the applicable Federal Acquisition Regulation and agency-specific supplemental regulations. As such, the use, duplication, disclosure, modification, and adaptation shall be subject to the restrictions and license terms set forth in the applicable Government contract, and, to the extent applicable by the terms of the Government contract, the additional rights set forth in FAR 52.227-19, Commercial Computer Software License (December 2007). Oracle USA, Inc., 500 Oracle Parkway, Redwood City, CA 94065.

This software or hardware is developed for general use in a variety of information management applications. It is not developed or intended for use in any inherently dangerous applications, including applications which may create a risk of personal injury. If you use this software or hardware in dangerous applications, then you shall be responsible to take all appropriate fail-safe, backup, redundancy, and other measures to ensure the safe use. Oracle Corporation and its affiliates disclaim any liability for any damages caused by use of this software or hardware in dangerous applications.

Oracle and Java are registered trademarks of Oracle and/or its affiliates. Other names may be trademarks of their respective owners.

AMD, Opteron, the AMD logo, and the AMD Opteron logo are trademarks or registered trademarks of Advanced Micro Devices. Intel and Intel Xeon are trademarks or registered trademarks of Intel Corporation. All SPARC trademarks are used under license and are trademarks or registered trademarks of SPARC International, Inc. UNIX is a registered trademark licensed through X/Open Company, Ltd.

This software or hardware and documentation may provide access to or information on content, products, and services from third parties. Oracle Corporation and its affiliates are not responsible for and expressly disclaim all warranties of any kind with respect to third-party content, products, and services. Oracle Corporation and its affiliates will not be responsible for any loss, costs, or damages incurred due to your access to or use of third-party content, products, or services.

Version 20100713142149.

Contents

Chapter 1

Introduction	28
Overview	28
JMS Functionality	28
Publish-and-Subscribe (Pub/Sub)	29
Point-to-Point (P2P)	29
Request/Reply	29
Message Selector	30
Additional Supported Resources	30
Multiplexer e*Way Functionality	30
Request/Reply	31
Send-Only	31
Receive	32
z/OS 1.2, 1.3, 1.4 client onlySystem Requirements	32
For Using Java Message Service APIs	32
For Using Java Message Service COM+ APIs	32
For Using Java Message Service C/C++ and RPG APIs	32
For Using Multiplexer e*Gate APIs	32
With the many compilers available, it is possible that some will not be compatible with the e*Gate environment.OS/390 and z/OS System Requirements	33
External System Requirements for OS/390	34
For Using CICS	34
For Using IMS	34
For Using Batch	35

Chapter 2

Installing the e*Gate API Kit	36
Installing the e*Gate API Kit on Windows	36
To install the e*Gate API Kit	36
Installing the e*Gate API Kit on UNIX	37
Pre-installation	37
Installing the e*Gate API Kit	37
Installing the e*Gate API Kit on UNIX	38
Pre-installation	38
Installing the e*Gate API Kit	38

Installing the e*Gate API Kit Connecting to an OS/400	39
Pre-installation	39
Installing the e*Gate API Kit	40
Installing the e*Gate API Kit on an OS/390 or z/OS	41
Installing from Tape	41
Installing from CD	42
Link Editing the COBOL API Object Modules	43
Linking the COBOL API Load Models	44
Verifying the CICS Transaction Server Environment for e*Gate	45
Directories and Files Created by the Installation	45
Files and Directories - Windows	45
Files and Directories - Solaris	47
Files and Directories - Other Systems	56
Configuring Perl	57

Chapter 3

Configuring the Message Service	59
Configuring the Message Service Clients	59
Java Client	60
Setting up the Java Client	61
COM+ Client	61
Setting up the COM+ Client	61
Viewing the Message Service COM+ APIs	61
Compensating Resource Manager (CRM)	62
Configuring the Message Server	72
Considerations	72

Chapter 4

Implementing JMS	73
Implementing JMS Models in Java or COM+	73
Considerations	73
Message Overview	74
Message Structure	74
Message Header Fields	74
Message Properties	76
Message Body (Payload)	76
Sample Code for Using JMS in Java and COM+	76
The Publish/Subscribe Model	77
Java Publish	78
Java Subscribe	79
COM VB Publish/Subscribe	81
ASP Publish	82
The Point-to-Point Model	83
Java Point-to-Point Sender	84
Java Point-to-Point Receiver	85

COM VB Point-to-Point	87
The Request-Reply Model	89
Java Request-Reply	89
Java TopicRequestor	90
Java QueueRequestor	91
COM VB TopicRequestor	94
COM VB QueueRequestor	94
JNDI	95
Initial Context	98
Naming Operations	98
JNDI Samples	98
The Message Selector	101
Message Selector Syntax	101
Java Message Selector Publisher	102
Java Message Selector Subscriber	104
COM VB Message Selector	105
XA Sample	106
Java XA Publisher	106
Java XA Subscriber	108
COM VB XA Sample	111
The Compensating Resource Manager	112
Sample Schema Implementation	119
e*Gate Sample JMS Schema Overview	120
Oracle SeeBeyond JMS IQ Manager	121
Event Type	121
Event Type Definition	121
JMS e*Way Connections	121
Java Collaboration Rules	121
Multi-Mode e*Way	123
Java Collaboration	123
Executing the Schema	124
Implementing JMS Models in C	125
Wrapper Functions for C	125
Sample Code for Using JMS in C	125
Publish/Subscribe Messaging Using C	126
Queue Messaging (Sending/Receiving) Using C	131
Request-Reply Messaging Using C	136
Message Selector Using C	140
Publish/Subscribe Messaging Using XA in C	145
Implementing JMS Models in C++	153
Sample Code for Using JMS in C++	153
Publish/Subscribe Messaging Using C++	154
Queue Messaging (Sending/Receiving) Using C++	158
Request-Reply Messaging Using C++	162
Message Selector Using C++	164
XA Publish/Subscribe Messaging For JMS Using C++	169
Sample Code for Using JMS in C++	175
Publish/Subscribe Messaging Using C++	175
Queue Messaging (Sending/Receiving) Using C++	179
Request-Reply Messaging Using C++	183

Message Selector Using C++	186
XA Publish/Subscribe Messaging For JMS Using C++	190

Chapter 5

Client Libraries for the JMS API 197

The Standard Specification for the JMS API 197

Supported Java Classes for JMS 197

<code>com.seebeyond.jms.client.STCTopicRequestor</code>	197
The STCTopicRequestor Method	198
The Request Method	198
The Request Method	198
The Close Method	198
<code>com.seebeyond.jms.STCQueueRequestor</code>	198
The STCQueueRequestor Method	199
The Request Method	199
The Request Method	199
<code>class javax.jms.JMSException</code>	199
The JMSException Method	200
The JMSException Method	200
The getErrorCode Method	200
The getLinkedException	200
The setLinkedException	200
<code>class javax.jms.IllegalStateException</code>	200
The IllegalStateException Method	201
The IllegalStateException Method	201
<code>class javax.jms.InvalidClientIDException</code>	201
The InvalidClientIDException	201
The InvalidClientIDException	201
<code>class javax.jms.InvalidDestinationException</code>	201
The InvalidDestinationException Method	202
The InvalidDestinationException Method	202
<code>class javax.jms.InvalidSelectorException</code>	202
The InvalidSelectorException Method	202
The InvalidSelectorException Method	202
<code>class javax.jms.JMSSecurityException</code>	203
The JMSSecurityException Method	203
The JMSSecurityException Method	203
<code>class javax.jms.MessageEOFException</code>	203
The MessageEOFException Method	203
The MessageEOFException	204
<code>class javax.jms.MessageFormatException</code>	204
The MessageFormatException Method	204
The MessageFormatException	204
<code>class javax.jms.MessageNotReadableException</code>	204
The MessageNotReadableException Method	205
The MessageNotReadable Method	205
<code>class javax.jms.MessageNotWriteableException</code>	205
The MessageNotWriteableException Method	205
The MessageNotWriteableException Method	205
<code>class javax.jms.ResourceAllocationException</code>	206
The ResourceAllocationException Method	206
The ResourceAllocationException Method	206
<code>class javax.jms.TransactionInProgressException</code>	206
The TransactionInProgressException Method	206
The TransactionInProgressException Method	207
<code>class javax.jms.TransactionRolledBackException</code>	207

The TransactionRolledBackException Method	207
The TransactionRolledBackException Method	207
Unsupported JMS Classes	208
Supported JMS Interfaces	208
interface javax.jms.Connection	208
The close Method	208
The getClientID Method	208
The getExceptionListener Method	208
The getMetaData Method	209
The setClientID Method	209
The setExceptionListener Method	209
The Start Method	209
The Stop Method	209
interface javax.jms.QueueConnection	209
The createQueueSession Method	210
interface javax.jms.XAQueueConnection	210
createXAQueueSession	210
createQueueSession	210
interface javax.jms.TopicConnection	211
The createTopicSession Method	211
interface javax.jms.XATopicConnection	211
The createTopicSession Method	212
createXATopicSession	212
interface javax.jms.ConnectionFactory	212
interface javax.jms.QueueConnectionFactory	213
The createQueueConnection Method	213
The createQueueConnection Method	213
interface javax.jms.XAConnectionFactory	214
interface javax.jms.TopicConnectionFactory	214
The createTopicConnection Method	214
The createTopicConnection Method	214
interface javax.jms.XATopicConnectionFactory	215
The createXATopicConnection Method	215
The createXATopicConnection Method	215
interface javax.jms.ConnectionMetaData	215
The getJMSVersion Method	215
The getJMSMajorVersion Method	216
The getJMSMinorVersion Method	216
The getJMSProviderName Method	216
The getProviderVersion Method	216
The getProviderMajorVersion Method	216
The getProviderMinorVersion Method	217
The getJMSXPropertyNames Method	217
interface javax.jms.DeliveryMode	217
interface javax.jms.Destination	218
interface javax.jms.Queue	218
The getQueueName Method	218
The toString Method	218
interface javax.jms.TemporaryQueue	218
The delete Method	218
interface javax.jms.Topic	219
The getTopicName Method	219
The toString Method	219
interface javax.jms.TemporaryTopic	219
The delete Method	219
interface javax.jms.ExceptionListener	220
The onException Method	220
interface javax.jms.Message	220
The getJMSMessageID Method	226

The setJMSMessageID Method	226
The getJMSTimestamp Method	227
The setJMSTimestamp Method	227
The getJMSCorrelationIDAsBytes Method	227
The setJMSCorrelationIDAsBytes Method	227
The setJMSCorrelationID Method	227
The getJMSCorrelationID Method	228
The getJMSReplyTo Method	228
The setJMSReplyTo Method	228
The getJMSDestination Method	229
The setJMSDestination Method	229
The getJMSDeliveryMode Method	229
The setJMSDeliveryMode Method	229
The getJMSRedelivered Method	230
The setJMSRedelivered Method	230
The getJMSType Method	230
The setJMSType Method	230
The setJMSType Method	231
The getJMSExpiration Method	231
The setJMSExpiration Method	231
The getJMSPriority Method	231
The setJMSPriority Method	232
The clearProperties Method	232
The propertyExists Method	232
The getBooleanProperty Method	232
The getByteProperty Method	233
The getIntProperty Method	233
The getLongProperty Method	233
The getFloatProperty Method	234
The getDoubleProperty Method	234
The getStringProperty Method	234
The getObjectProperty Method	235
The getPropertyNames Method	235
The setBooleanProperty Method	235
The setByteProperty Method	235
The setShortProperty Method	236
The setIntProperty Method	236
The setLongProperty Method	236
The setFloatProperty Method	237
The setDoubleProperty Method	237
The setStringProperty Method	237
The setObjectProperty Method	238
The acknowledge Method	238
The clearBody Method	238
interface javax.jms.BytesMessage	239
The readBoolean Method	239
The readByte Method	239
The readUnsignedByte Method	239
The readShort Method	239
The readUnsignedShort Method	240
The readChar Method	240
The readInt Method	240
The readLong Method	240
The readFloat Method	241
The readDouble Method	241
The readUTF Method	241
The readBytes Method	241
The readBytes Method	242
The writeBoolean Method	242
The writeByte Method	243
The writeShort Method	243
The writeChar Method	243
The writeInt Method	243
The writeLong Method	244

The writeFloat Method	244
The writeDouble Method	244
The writeUTF Method	245
The writeBytes Method	245
The writeBytes Method	245
The writeObject Method	246
The reset Method	246
interface javax.jms.MapMessage	246
The getBoolean Method	247
The getByte Method	247
The getShort Method	247
The getChar Method	247
The getInt Method	248
The getLong Method	248
The getFloat Method	248
The getDouble Method	249
The getString Method	249
The getBytes Method	249
The getObject Method	249
The getMapNames Method	250
The setBoolean Method	250
The setByte Method	250
The setShort Method	250
The setChar Method	251
The setInt Method	251
The setLong Method	251
The setFloat Method	252
The setDouble Method	252
The setString Method	252
The setBytes Method	253
The setBytes Method	253
The setObject Method	253
The itemExists Method	254
interface javax.jms.ObjectMessage	254
The setObject Method	254
The getObject Method	255
interface javax.jms.StreamMessage	255
The readBoolean Method	255
The readByte Method	255
The readShort Method	256
The readChar Method	256
The readInt Method	256
The readLong Method	257
The readFloat Method	257
The readDouble Method	257
The readString Method	257
The readBytes Method	258
The readObject Method	258
The writeBoolean Method	259
The writeByte Method	259
The writeShort Method	259
The writeChar Method	259
The writeInt Method	260
The writeLong Method	260
The writeFloat Method	260
The writeDouble Method	260
The writeString Method	261
The writeBytes Method	261
The writeBytes Method	261
The writeObject Method	262
The reset Method	262
interface javax.jms.TextMessage	262
The getText Method	263
The setText Method	263

interface javax.jms.MessageConsumer	263
The getMessageSelector Method	263
The getMessageListener Method	264
The setMessageListener Method	264
The receive Method	264
The receive Method	264
The receiveNoWait Method	265
The close Method	265
interface javax.jms.QueueReceiver	265
The getQueue Method	265
interface javax.jms.TopicSubscriber	265
The getTopic Method	266
The getNoLocal Method	266
interface javax.jms.MessageListener	267
The onMessage Method	267
interface javax.jms.MessageProducer	267
The setDisableMessageID Method	267
The getDisableMessageID Method	268
The setDisableMessageTimestamp Method	268
The getDisableMessageTimestamp Method	268
The setDeliveryMode Method	269
The getDeliveryMode Method	269
The setPriority Method	269
The getPriority Method	269
The setTimeToLive Method	270
The getTimeToLive Method	270
The close Method	270
interface javax.jms.QueueSender	270
The getQueue Method	270
The send Method	271
The send Method	271
The send Method	271
The send Method	272
interface javax.jms.TopicPublisher	272
The getTopic Method	273
The publish Method	273
The publish Method	273
The publish Method	274
The publish Method	274
interface java.lang.Runnable	275
interface javax.jms.Session	275
The createBytesMessage Method	276
The createMapMessage Method	276
The createMessage Method	277
The createObjectMessage Method	277
The createObjectMessage Method	277
The createStreamMessage Method	277
The createTextMessage Method	277
The createTextMessage Method	278
The getTransacted Method	278
The commit Method	278
The rollback Method	278
The close Method	279
The recover Method	279
The getMessageListener Method	279
The setMessageListener Method	280
interface javax.jms.QueueSession	280
The createQueue Method	280
The createReceiver Method	281
The createReceiver Method	281
The createSender Method	281
The createTemporaryQueue Method	282

interface javax.jms.TopicSession	282
The createTopic Method	282
The createSubscriber Method	283
The createSubscriber Method	283
The createDurableSubscriber Method	284
The createDurableSubscriber Method	284
The createPublisherMethod	285
The createTemporaryTopic Method	285
The unsubscribe Method	286
interface javax.jms.XASession	286
The getXAResource Method	286
The getTransacted Method	286
The commit Method	287
The rollback Method	287
interface javax.jms.XAQueueSession	287
The getQueueSession Method	287
interface javax.jms.XATopicSession	287
The getTopicSession Method	288
interface javax.jms.XAConnection	288
interface javax.jms.XAQueueConnection	288
The createXAQueueSession Method	288
The createQueueSession Method	288
interface javax.jms.XATopicConnection	289
The createXATopicSession Method	289
The createTopicSession Method	289
interface javax.jms.XAConnectionFactory	289
interface javax.jms.XAQueueConnectionFactory	289
The createXAQueueConnection Method	290
The createXAQueueConnection Method	290
interface javax.jms.XATopicConnectionFactory	290
Unsupported Java JMS Classes	290
Unsupported Java JMS Interfaces	291
Unsupported JMS Methods	291
The Message Service COM+ APIs	291
Supported Java Message Service (JMS) Classes for COM+	291
The BytesMessage Object	292
The Acknowledge Method	292
The ClearBody Method	292
The ClearProperties Method	292
The GetProperty Method	292
The PropertyExists Method	293
The ReadBoolean Method	293
The ReadByte Method	293
The ReadBytes Method	293
The ReadChar Method	293
The ReadDouble Method	293
The ReadFloat Method	293
The ReadInt Method	294
The ReadLong Method	294
The ReadShort Method	294
The ReadUnsignedByte Method	294
The ReadUnsignedShort Method	294
The ReadUTF Method	294
The Reset Method	294
The SetProperty Method	294
The WriteBoolean Method	295
The WriteByte Method	295
The WriteBytes Method	295

The WriteChar Method	295
The WriteDouble Method	296
The WriteFloat Method	296
The WriteInt Method	296
The WriteLong Method	296
The WriteObject Method	296
The WriteShort Method	297
The WriteUTF Method	297
Properties of the BytesMessage Object	297
The CorrelationID Property	297
The CorrelationIDAsBytes Property	297
The DeliveryMode Property	297
The Destination Property	298
The Expiration Property	298
The MessageID Property	298
The Priority Property	298
The Redelivered Property	298
The ReplyTo Property	298
The Timestamp Property	299
The Type Property	299
The Connection Object	299
The Start Method	299
The Stop Method	299
Properties of the Connection Object	299
The ClientID Property	299
The MetaData Property	299
The ConnectionFactory Object	300
Properties of the ConnectionFactory Object	300
The HostName Property	300
The Port Property	300
The PortOffset Property	300
The Connection MetaData Object	300
The MapMessage Object	300
The Acknowledge Method	300
The ClearBody Method	301
The ClearProperties Method	301
The GetBoolean Method	301
The GetByte Method	301
The GetBytes Methods	301
The GetChar Method	301
The GetDouble Method	302
The GetFloat Method	302
The GetInt Method	302
The GetLong Method	302
The GetObject Method	302
The GetProperty Method	303
The GetShort Method	303
The GetString Method	303
The ItemExists Method	303
The PropertyExists Method	303
The SetBoolean Method	304
The SetByte Method	304
The SetBytes Method	304
The SetChar Method	304
The SetDouble Method	305
The SetFloat Methods	305
The SetInt Method	305
The SetLong Method	305
The SetObject Method	305
The SetProperty Method	306
The SetShort Method	306
The SetString Method	306
Properties of the MapMessage Object	306

The CorrelationID Property	306
The CorrelationIDAsBytes Property	306
The DeliveryMode Property	307
The Destination Property	307
The Expiration Property	307
The MapNames Property	307
The MessageID Property	307
The Priority Property	308
The Redelivered Property	308
The ReplyTo Property	308
The Timestamp Property	308
The Type Property	308
The Message Object	308
The Acknowledge Method	308
The ClearBody Method	309
The ClearProperties Method	309
The GetProperty Method	309
The PropertyExists Method	309
The SetProperty Method	309
Properties of the Message Object	310
The CorrelationID Property	310
The CorrelationIDAsBytes Property	310
The DeliveryMode Property	310
The Destination Property	310
The Expiration Property	310
The MessageID Property	311
The ReplyTo Property	311
The Timestamp Property	311
The Type Property	311
The MessageConsumer Object	311
The Close Method	311
The Receive Message Method	312
The ReceiveNoWait Method	312
Properties of the MessageConsumer Object	312
The MessageListener Property	312
The MessageSelector Property	312
The MessageListener Object	312
The OnMessage Property	312
The MessageProducer Object	312
Properties of the MessageProducer Object	313
The DeliveryMode Property	313
The DisableMessageID Property	313
The DisableMessageTimestamp Property	313
The Priority Method	313
The Queue Object	314
The ToString Method	314
Properties of the Queue Object	314
The QueueName Property	314
The QueueBrowser Object	314
The QueueConnection Object	314
The CreateQueueSession Method	314
The Start Method	315
The Stop Method	315
Properties of QueueConnection Object	315
The ClientID Property	315
The MetaData Property	315
The QueueConnectionFactory Object	315
The CreateQueueConnection Method	316
Properties of the QueueConnectionFactory Object	316
The HostName Property	316
The Port Property	316
The PortOffset Property	316

The QueueReceiver Object	316
The Close Method	316
The Receive Method	316
The ReceiveNoWait Method	317
Properties of the QueueReceiver Object	317
The MessageListener Property	317
The MessageSelector Property	317
The Queue Property	317
The QueueRequestor Object	317
The Create Method	317
The Request Method	318
The QueueSender Object	318
The Send Method	318
Properties of the QueueSender Object	318
The DeliveryMode Property	318
The DisableMessageID Property	319
The DisableMessageTimestamp Property	319
The Priority Property	319
The Queue Property	319
The TimeToLive Property	319
The QueueSession Object	319
The Commit Method	320
The CreateBrowser Method	320
The CreateBytesMessage Method	320
The CreateMapMessage Method	320
The CreateMessage Method	320
The CreateQueue Method	320
The CreateReceiver Method	320
The CreateSender Method	321
The CreateStreamMessage Method	321
The CreateTemporaryQueue Method	321
The CreateTextMessage Method	321
The Recover Method	321
The Rollback Method	322
The Run Method	322
Properties of the QueueSender Object	322
The MessageListener Property	322
The Transacted Property	322
The Session Object	322
The Commit Method	322
The CreateBytesMessage Method	322
The CreateMapMessage Method	322
The CreateMessage Method	323
The CreateStreamMessage Method	323
The CreateTextMessage Method	323
The Recover Method	323
The Rollback Method	323
The Run Method	323
Properties of the Session Object	323
The MessageListener Property	323
The Transacted Property	324
The StreamMessage Object	324
The Acknowledge Method	324
The ClearBody Method	324
The ClearProperties Method	324
The GetProperty Method	324
The PropertyExists Method	324
The ReadBoolean Method	325
The ReadByte Method	325
The ReadBytes Method	325
The ReadChar Method	325
The ReadDouble Method	325

The ReadFloat Method	325
The ReadInt Method	326
The ReadLong Method	326
The ReadObject Method	326
The ReadShort Method	326
The ReadString Method	326
The Reset Method	326
The SetProperty Method	326
The WriteByte Method	327
The WriteBytes Method	327
The WriteChar Method	327
The WriteDouble Method	327
The WriteFloat Method	328
The WriteInt Method	328
The WriteLong Method	328
The WriteObject Method	328
The WriteShort Method	328
The WriteString Method	328
Properties of the StreamMessage Object	329
The CorrelationID Property	329
The CorrelationIDAsBytes Property	329
The DeliveryMode Property	329
The Destination Property	329
The Expiration Property	330
The MessageID Property	330
The Priority Property	330
The Redelivered Property	330
The ReplyTo Property	330
The Timestamp Property	330
The Type Property	331
The TemporaryQueue Object	331
The Delete Method	331
The ToString Method	331
Properties of the TemporaryQueue Object	331
The QueueName Property	331
The TemporaryTopic Object	331
The Delete Method	331
The ToString Method	332
Properties of the TemporaryTopic Object	332
The TopicName Property	332
The TextMessage Object	332
The Acknowledge Method	332
The ClearBody Method	332
The ClearProperties Method	332
The GetProperty Method	332
The PropertyExists Method	333
The SetProperty Method	333
Properties of the Message Object	333
The CorrelationID Property	333
The CorrelationIDAsBytes Property	333
The DeliveryMode Property	333
The Destination Property	334
The Expiration Property	334
The MessageID Property	334
The Priority Property	334
The Redelivered Property	335
The ReplyTo Property	335
The Text Property	335
The Timestamp Property	335
The Type Property	335
The Topic Object	335
The ToString Method	336
Properties of the Topic Object	336

The TopicName Property	336
The TopicConnection Object	336
The CreateTopicSession Method	336
The Start Method	336
The Stop Method	337
Properties of the TopicConnection	337
The ClientID Property	337
The MetaData Property	337
The TopicConnectionFactory Object	337
The CreateTopicConnection Method	337
Properties of the TopicConnectionFactory	337
The HostName Property	337
The Port Property	337
The PortOffset Property	338
The TopicPublisher Object	338
The Publish Method	338
Properties of TopicPublisher	338
The DeliveryMode Property	338
The DisableMessageID Property	339
The DisableMessageTimestamp Property	339
The Priority Property	339
The TimeToLive Property	339
The Topic Property	340
The TopicRequestor Object	340
The Create Method	340
The Request Method	340
The TopicSession Object	340
The Commit Method	340
The CreateBytesMessage Method	341
The CreateDurableSubscriber Method	341
The CreateMapMessage Method	341
The CreateMessage Method	341
The CreatePublisher Method	341
The CreateStreamMessage Method	342
The CreateSubscriber Method	342
The CreateTemporaryTopic Method	342
The CreateTextMessage Method	342
The CreateTopic Method	342
The Recover Method	343
The Rollback Method	343
The Run Method	343
The Unsubscribe Method	343
Properties of the TopicSession Object	343
The MessageListener Property	343
The Transacted Property	343
The TopicSubscriber Object	343
The Close Method	344
The Receive Method	344
The ReceiveNoWait Method	344
Properties of the TopicSubscriber Object	344
The MessageListener Property	344
The MessageSelector Property	344
The NoLocal Property	344
The Topic Property	344
The XAQueueConnection Object	345
The CreateQueueSession Method	345
The CreateXAQueueSession Method	345
The Start Method	345
The Stop Method	346
Properties of XAQueueConnection Object	346
The ClientID Property	346
The MetaData Property	346

The XAQueueConnectionFactory Object	346
The CreateQueueConnection Method	346
The CreateXAQueueConnection Method	346
Properties of the QueueConnectionFactory Object	346
The HostName Property	346
The Port Property	347
The PortOffset Property	347
The XAQueueSession Object	347
The Commit Method	347
The CreateBytesMessage Method	347
The CreateMapMessage Method	347
The CreateMessage Method	347
The CreateStreamMessage Method	347
The CreateTextMessage Method	348
The Recover Method	348
The Rollback Method	348
The Run Method	348
Properties of the QueueSender Object	348
The MessageListener Property	348
The QueueSession Property	348
The Transacted Property	348
The XASession Object	349
The Commit Method	349
The CreateBytesMessage Method	349
The CreateMapMessage Method	349
The CreateMessage Method	349
The CreateStreamMessage Method	349
The CreateTextMessage Method	349
The Recover Method	350
The Rollback Method	350
The Run Method	350
Properties of the Session Object	350
The MessageListener Property	350
The Transacted Property	350
The XATopicConnection Object	350
The CreateTopicSession Method	350
The Start Method	351
The Stop Method	351
Properties of the TopicConnection	351
The ClientID Property	351
The MetaData Property	351
The XATopicConnectionFactory Object	352
The CreateTopicConnection Method	352
The CreateXATopicConnection Method	352
Properties of the TopicConnectionFactory	352
The HostName Property	352
The Port Property	352
The PortOffset Property	352
The XATopicSession Object	352
The Commit Method	353
The CreateBytesMessage Method	353
The CreateMapMessage Method	353
The CreateMessage Method	353
The CreateStreamMessage Method	353
The CreateTextMessage Method	353
The Recover Method	353
The Rollback Method	354
The Run Method	354
Properties of the TopicSession Object	354
The MessageListener Property	354
The TopicSession Property	354
The Transacted Property	354

The C API for JMS	355
Architectural Overview	356
Structures	357
Constants	357
DeliveryMode Constants	358
DestinationType Constants	358
MessageType Constants	358
Session Constants	358
Transacted Constants	359
Miscellaneous Constants Setting Message Class Defaults	360
Other Miscellaneous Constants	360
Interfaces	360
The Message Interface	361
Acknowledge	362
ClearBody	363
ClearProperties	363
PropertyExists	364
GetBooleanProperty	364
GetByteProperty	365
GetDoubleProperty	365
GetFloatProperty	366
GetIntProperty	366
GetLongProperty	367
GetShortProperty	367
GetStringProperty	368
SetBooleanProperty	368
SetByteProperty	369
SetDoubleProperty	369
SetFloatProperty	370
SetIntProperty	370
SetLongProperty	371
SetShortProperty	371
SetStringProperty	372
GetJMSCorrelationID	372
GetJMSCorrelationIDAsBytes	372
GetJMSDeliveryMode	373
GetJMSExpiration	373
GetJMSMessageID	374
GetJMSPriority	374
GetJMSRedelivered	375
GetJMSReplyTo	375
GetJMSTimestamp	376
GetJMSType	376
SetJMSCorrelationID	377
SetJMSCorrelationIDAsBytes	377
SetJMSDeliveryMode	378
SetJMSExpiration	378
SetJMSMessageID	378
SetJMSPriority	379
SetJMSRedelivered	379
SetJMSReplyTo	380
SetJMSTimestamp	380
SetJMSType	381
The Extended Message Interface	381
GetMessageType	381
BytesMessage Methods	382
ReadBoolean	383
ReadByte	383
ReadBytes	384
ReadChar	384
ReadDouble	385
ReadFloat	385
ReadInt	386

ReadLong	386
ReadShort	386
ReadUnsignedByte	387
ReadUnsignedShort	387
ReadUTF	388
Reset	388
WriteBoolean	389
WriteByte	389
WriteBytes	390
WriteBytesEx	390
WriteChar	391
WriteDouble	391
WriteFloat	392
WriteInt	392
WriteLong	392
WriteShort	393
WriteUTF	393
TextMessage Methods	394
GetText	394
SetText	395
The QueueConnectionFactory Interface	395
CreateQueueConnectionFactory	395
CreateQueueConnection	396
The Connection Interface	396
ConnectionClose	397
ConnectionGetClientID	397
ConnectionSetClientID	397
ConnectionStart	398
ConnectionStop	398
ConnectionCreateQueueSession	399
ConnectionCreateTopicSession	399
The Session Interface	400
SessionClose	401
SessionCommit	401
SessionGetTransacted	402
SessionRecover	402
SessionRollback	402
SessionCreateBytesMessage	403
SessionCreateTextMessage	403
SessionCreateTextMessageEx	404
SessionCreateQueue	404
SessionCreateReceiver	405
SessionCreateReceiveMessageSelector	405
SessionCreateSender	406
SessionCreateTemporaryQueue	406
SessionCreateDurableSubscriber	407
SessionCreateDurableSubscriberMessageSelector	408
SessionCreatePublisher	408
SessionCreateSubscriber	409
SessionCreateSubscriberMessageSelector	409
SessionCreateTemporaryTopic	410
SessionCreateTopic	411
SessionUnsubscribe	411
The TopicConnectionFactory Interface	412
CreateTopicConnectionFactory	412
CreateTopicConnection	412
The Destination Interface	413
GetDestinationName	413
SetDestinationName	414
DestinationToString	414
DeleteDestination	414
The QueueReceiver Interface	415
QueueReceiverClose	415

QueueReceiverGetMessageSelector	416
QueueReceiverReceive	416
QueueReceiverReceiveTimeout	417
QueueReceiverReceiveNoWait	417
QueueReceiverGetQueue	417
The TopicSubscriber Interface	418
TopicSubscriberClose	418
TopicSubscriberGetMessageSelector	419
TopicSubscriberGetNoLocal	419
TopicSubscriberGetTopic	420
TopicSubscriberReceive	420
TopicSubscriberReceiveTimeout	420
TopicSubscriberReceiveNoWait	421
The QueueSender Interface	421
QueueSenderClose	422
QueueSenderGetDeliveryMode	423
QueueSenderGetDisableMessageID	423
QueueSenderGetDisableMessageTimestamp	424
QueueSenderGetJMS_ProducerID	424
QueueSenderGetPriority	424
QueueSenderGetQueue	425
QueueSenderGetTimeToLive	425
QueueSenderSend	426
QueueSenderSendEx	426
QueueSenderSendToQueue	427
QueueSenderSendToQueueEx	428
QueueSenderSetDeliveryMode	429
QueueSenderSetDisableMessageID	429
QueueSenderSetDisableMessageTimestamp	430
QueueSenderSetJMS_ProducerID	430
QueueSenderSetPriority	431
QueueSenderSetTimeToLive	431
The TopicPublisher Interface	432
TopicPublisherClose	432
TopicPublisherGetDeliveryMode	433
TopicPublisherGetDisableMessageID	433
TopicPublisherGetDisableMessageTimestamp	434
TopicPublisherGetJMS_ProducerID	434
TopicPublisherGetPriority	434
TopicPublisherGetTimeToLive	435
TopicPublisherGetTopic	435
TopicPublisherPublish	436
TopicPublisherPublishEx	436
TopicPublisherPublishToTopic	437
TopicPublisherPublishToTopicEx	438
TopicPublisherSetDeliveryMode	438
TopicPublisherSetDisableMessageID	439
TopicPublisherSetDisableMessageTimestamp	439
TopicPublisherSetJMS_ProducerID	440
TopicPublisherSetPriority	440
TopicPublisherSetTimeToLive	441
The TopicRequestor Interface	442
CreateTopicRequestor	442
TopicRequestorRequest	442
TopicRequestorRequestTimeout	443
TopicRequestorClose	443
The QueueRequestor Interface	444
CreateQueueRequestor	444
QueueRequestorClose	445
QueueRequestorRequest	445
QueueRequestorRequestTimeout	446
Destructor Methods	446
DeleteQueueConnectionFactory	447

DeleteQueueConnection	447
DeleteQueueReceiver	448
DeleteQueueRequestor	448
DeleteQueueSender	448
DeleteQueueSession	449
DeleteTopicConnectionFactory	449
DeleteTopicConnection	450
DeleteTopicSession	450
DeleteTopicSubscriber	450
DeleteTopicRequestor	451
DeleteTopicPublisher	451
DeleteMessage	452
The WString Helper Interface	452
CharToWString	452
DeleteWString	453
WStringToChar	453
The WStringList Helper Interface	454
DeleteWStringList	454
GetPropertyNames	454
Error Codes and Messages in the C API for JMS	455
Differences Between JMS Java API and JMS C API	456
Differences Between Java and C in the BytesMessage Interface	456
Differences Between Java and C in the MapMessage Interface	456
Differences Between Java and C in the MessageProducer Interface	456
Differences Between Java and C in Error Handling	456
The Supported C++ APIs for Oracle Seebeyond JMS	457
The C++ API for Oracle Seebeyond JMS	458
The Message Interface for JMS in C++	458
acknowledge	459
clearBody	459
clearProperties	460
propertyExists	460
getBooleanProperty	460
getByteProperty	461
getDoubleProperty	461
getFloatProperty	461
getIntProperty	462
getLongProperty	462
getPropertyNames	463
getShortProperty	463
getStringProperty	463
setBooleanProperty	464
setByteProperty	464
setDoubleProperty	465
setFloatProperty	465
setIntProperty	465
setLongProperty	466
setObjectProperty	466
setShortProperty	467
setStringProperty	467
propertyExists	467
getBooleanProperty	468
getByteProperty	468
getDoubleProperty	469
getFloatProperty	469
getIntProperty	469
getLongProperty	470
getObjectProperty	470
getShortProperty	471
getStringProperty	471
setBooleanProperty	471

setByteProperty	472
setDoubleProperty	472
setFloatProperty	473
setIntProperty	473
setLongProperty	473
setObjectProperty	474
setShortProperty	474
setStringProperty	475
getJMSCorrelationID	475
getJMSCorrelationIDAsBytes	475
getJMSDeliveryMode	475
getJMSExpiration	476
getJMSMessageID	476
getJMSPriority	476
getJMSRedelivered	477
getJMSReplyTo	477
getJMSTimestamp	477
getJMSType	477
setJMSCorrelationID	478
setJMSCorrelationIDAsBytes	478
setJMSDeliveryMode	478
setJMSExpiration	479
setJMSMessageID	479
setJMSPriority	480
setJMSRedelivered	480
setJMSReplyTo	480
setJMSTimestamp	481
setJMSType	481
setJMSCorrelationIDAsBytes	481
setJMSMessageID	482
setJMSType	482
The BytesMessage Interface for JMS in C++	483
readBoolean	483
readByte	483
readChar	484
readDouble	484
readFloat	484
readInt	484
readLong	485
readShort	485
readUnsignedByte	485
readUnsignedShort	485
readUTF	486
reset	486
writeBoolean	486
writeByte	487
writeBytes	487
writeBytes	487
writeChar	488
writeDouble	488
writeFloat	489
writeInt	489
writelnLong	489
writeShort	490
None.The MapMessage Class	490
getBoolean	490
getByte	491
getBytes	491
getBytes	491
getChar	492
getDouble	492
getFloat	493
getInt	493

getLong	493
The GetObject Method	494
The GetProperty Method	494
The GetShort Method	494
The GetString Method	494
The ItemExists Method	494
The PropertyExists Method	495
The SetBoolean Method	495
The SetByte Method	495
The SetBytes Method	495
The SetChar Method	496
The SetDouble Method	496
The SetFloat Methods	496
The SetInt Method	496
The SetLong Method	497
The SetObject Method	497
The SetProperty Method	497
The SetShort Method	497
The SetString Method	497
The SetString Method	498
The TextMessage Class	498
getText	498
setText	498
setText	499
The Connection Interface for JMS in C++	499
close	499
getClientID	500
setClientID	500
start	500
stop	500
The QueueConnection Interface for JMS in C++	501
createQueueSession	501
The Session Interface for JMS in C++	501
close	502
commit	502
getTransacted	502
recover	502
rollback	503
bytesMessage	503
createTextMessage	503
createTextMessage	504
The TopicConnection Interface for JMS in C++	504
createTopicSession	504
close	505
getClientID	505
setClientID	506
setClientID	506
getExceptionListener	506
The QueueConnectionFactory Interface for JMS in C++	507
createQueueConnection	507
createQueueConnection	507
The TopicConnectionFactory Interface for JMS in C++	507
createTopicConnectionFactory	508
The ExceptionListener Interface for JMS in C++	508
onException	508
The DeliveryMode Interface for JMS in C++	509
The Queue Interface for JMS in C++	509
getQueueName	510
toString	510
The TemporaryQueue Interface for JMS in C++	510
Delete	510
The Topic Interface for JMS in C++	511

getTopicName	511
toString	511
The TemporaryTopic Interface for JMS in C++	512
Delete	512
The MessageProducer Interface for JMS in C++	512
close	513
getDeliveryMode	513
getDisableMessageID	513
getDisableMessageTimestamp	513
getJMS_ProducerID	514
getPriority	514
getTimeToLive	514
setDeliveryMode	514
setDisableMessageID	515
setDisableMessageTimestamp	515
setJMS_ProducerID	516
setPriority	516
setTimeToLive	516
The QueueSender Interface for JMS in C++	517
send	517
send	518
send	518
send	518
send	519
send	519
send	520
send	520
The TopicPublisher Interface	521
getTopic	521
publish	522
publish	522
publish	522
publish	523
publish	523
publish	524
publish	524
publish	525
The QueueSession Interface for JMS in C++	525
createQueue	525
createReceiver	526
createReceiver	526
createSender	527
createTemporaryQueue	527
The TopicSession Interface for JMS in C++	527
createDurableSubscriber	528
createDurableSubscriber	528
createPublisher	529
createSubscriber	529
createSubscriber	530
createTemporaryTopic	530
createTopic	531
unsubscribe	531
The Xid Interface for JMS in C++	532
getBranchQualifier	532
getFormatId	532
getGlobalTransactionId	532
The XAResource Interface for JMS in C++	533
commit	533
Xid**recover	534
rollback	534
getTransactionTimeout	535
setTransactionTimeout	535
isSameRM	535

prepare	536
start	536
end	537
MSGSRVC_API *Lookup	537
*LookupQueueConnectionFactory	538
*LookupXAQueueConnectionFactory	538
*LookupQueue	539
*LookupTopicConnectionFactory	539
*LookupXATopicConnectionFactory	540
*LookupTopic	541
*CreateXid	541
*LookupXADataSource	542
*LookupQueueConnectionFactoryExt	542
*LookupXAQueueConnectionFactoryExt	543
*LookupXATopicQueueConnectionFactoryExt	544

Chapter 6

Configuring the Multiplexer e*Way 546

Configuring the Multiplexer Client	546
Before You Begin	546
Setting up the Multiplexer	546
Setting up the Muxpooler	547
Configuring the Multiplexer Server	547
Multiplexer e*Way Configuration Parameters	547
General Settings	548
Notes on Session ID, Instance ID, and Recovery ID	549

Chapter 7

Implementing the Multiplexer e*Way 550

Implementing the Multiplexer Models	550
Multiplexer Overview	550
Request Reply	550
Multiplexer Request/Reply Sample Schema	552
ETDs, Collaboration Rules, and the "Return Address" Header	554
Using the C APIs	555
Using the Java APIs	555
Using the ActiveX Control Within Visual Basic Applications	556
Using Perl APIs	556
Using the COBOL APIs	557
Sample Implementation	559

Chapter 8

Client Libraries for the Multiplexer e*Way 560

C API Function Prototypes	560
EWIPMP_Close	560

EWIPMP_Free	561
EWIPMP_Open	562
EWIPMP_Send	563
EWIPMP_Wait	563
COBOL APIs	565
Open	565
Send	566
Receive	567
Close	569
ActiveX APIs	569
Connect	570
Disconnect	570
LastErrorCode	571
LastErrorText	571
ReplyMessageAsArray	572
ReplyMessageAsString	572
ReplyMessageSize	572
Send	573
Wait	573
ActiveX Class ID	574
Java Methods	574
Defaults	574
connect	575
disconnect	575
getHost	576
getPort	576
getResponse	577
getResponseBytes	577
getSecondsToExpire	578
getSleepDuration	578
getTimeout	579
sendMessage	579
setDebug	580
setHost	581
setPort	581
setSecondsToExpire	582
setSleepDuration	582
setTimeout	583
com.stc.MUXPooler	584
Constructors	584
Methods	584
connect	585
disconnect	585
disconnect	585
getConnectionCount	586
getHost	586
getPort	587
getSecondsToExpire	587
getTimeout	587
resizeMUXPool	588
sendBytes	588
sendMessage	588
setConnectionCount	589
setHost	589
setPort	590
setSecondsToExpire	590
setTimeout	590
Perl Subroutines	591
Multiplexer_Close	591

Contents

Multiplexer_Free	592
Multiplexer_Init	592
Multiplexer_Open	593
Multiplexer_Send	594
Multiplexer_ToString	594
Multiplexer_Wait	595

Appendix A

Appendix **597**

Cobol API Return Codes 597

Cobol Error Return Codes 597

 TCP/IP for MVS Return Codes 598

 Sockets Extended Return Codes 606

Index **611**

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:

Java Message Service (JMS)

- Java
- COM +
- C/C++, RPG

Multiplexer (MUX) e*Way

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

1.1 Overview

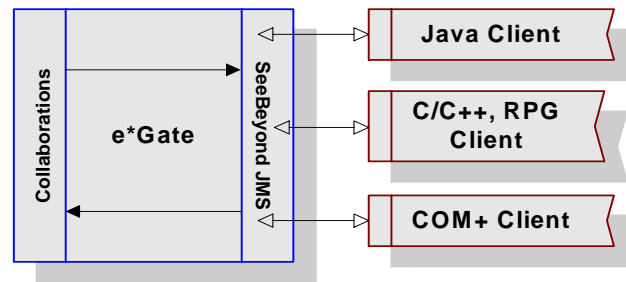
The e*Gate API Kit provides two distinct IQ delivery service mechanisms:

- ICAN Suite JMS
- Multiplexer e*Way

1.1.1. JMS Functionality

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

The basic JMS data flow is illustrated in [Figure 1 on page 29](#).

Figure 1 Basic Message Service Data Flow

Several common programming models are supported by the ICAN Suite JMS API, including the following:

- **Publish/subscribe (pub/sub)**
- **Point-to-point (P2P)**
- **Request/reply**
- **Message selector**

Each of these programming models are briefly described below.

Publish-and-Subscribe (Pub/Sub)

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

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

Point-to-Point (P2P)

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

Request/Reply

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

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

Message Selector

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

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

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

Additional Supported Resources

The ICAN Suite implementation of JMS supports certain external resources that you can use when developing your messaging systems, including:

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

These resources are described below.

Java Naming and Directory Interface (JNDI)

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

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

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

JNDI is included in the Java 2 SDK. To use the JNDI functionality, the JNDI classes are required, along with one or more service providers (such as, LDAP, CORBA, or RMI).

Compensating Resource Manager

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

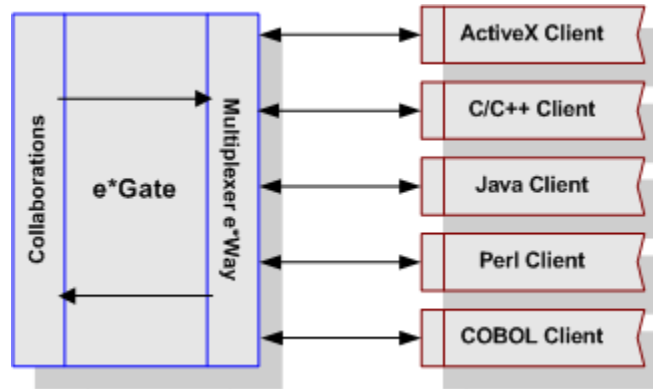
1.1.2. Multiplexer e*Way Functionality

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 back-end connectivity extends application, trading partner, and business process integration to the World Wide Web environment.

The multiplexer data flow is illustrated in Figure 2.

Figure 2 Multiplexer Data Flow



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

- **Request/reply**
- **Send-only**
- **Receive**

Each architecture type is described below.

Request/Reply

In the request/reply architecture, data is sent to the e*Gate system and a response is returned, as follows:

- 1 A client submits data (a **request**) to the e*Gate system.
- 2 The e*Gate system processes the data as required.
- 3 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

Send-only uses the same multiplexing e*Way component as request/reply. However, in this model, data is sent to the e*Gate system but no data is returned.

Receive

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

1.2 z/OS 1.2, 1.3, 1.4 client only System Requirements

All APIs require the Java 2 SDK. See the `readme.txt` file provided on the installation media for release information.

1.2.1. For Using Java Message Service APIs

To use the Java Message Service Java 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 (this is included with the Java 2 SDK)

1.2.2. 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, Microsoft Visual Basic

1.2.3. For Using Java Message Service C/C⁺⁺ and RPG APIs

To use the Java Message Service for C/C⁺⁺ and RPG with the UNIX, OS/390, z/OS, and the AS/400 samples included with this kit, you will need GNUMake. All of the samples were created using this program.

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

1.2.4. For Using Multiplexer e*Gate APIs

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

- A TCP/IP network connection
- 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	5.005_03 patch level ML10
HP-UX 11	5.005_03 patch level March 2002 Quality Pack
HP-UX 11i	5.005_03 patch level June 2002 Gold Base
Solaris	5.005_03 J2SE(5/22/02) Recommended (5/8/2002) 105181-31
Red Hat Linux	5.005_03 patch level None
Windows	5.003_03

HP-UX 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.

1.3 *With the many compilers available, it is possible that some will not be compatible with the e*Gate environment.* **OS/390 and z/OS System Requirements**

OS/390 and z/OS system requirements and installation procedures are covered in the *e*Gate Integrator Installation Guide*.

OS/390 and z/OS 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 or 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.

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

1.4 External System Requirements for OS/390

1.4.1. 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>

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 TS 1.x or higher
- ♦ 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 - UNIX System Services (VSS) installed, configured, and operational.

1.4.2. 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*"

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.

1.4.3. 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 Web site, 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

Note: Optional - Open Multiple Virtual System (OMVS) installed, configured, and operational.

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. The following platform types are discussed:

- [Installing the e*Gate API Kit on Windows](#) on page 36
- [Installing the e*Gate API Kit on UNIX](#) on page 37

For a list of the files the e*Gate API Kit will install on your system, see [“Directories and Files Created by the Installation” on page 45](#).

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 Installing the e*Gate API Kit on Windows

Note that you must have Administrator privileges to successfully install e*Gate, and that the e*Gate API Kit can only 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*.

2.1.1. To install the e*Gate API Kit

- 1 Log in as an Administrator on the workstation on which you want to install the Kit.
- 2 Exit all Windows programs, including any anti-virus applications.
- 3 Insert the e*Way installation CD-ROM into the CD-ROM drive.

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

The InstallShield setup application launches.

- 4 When the **Select Component Categories** window appears, select e*Ways and then click **Change**. On the **Select Sub-components** dialog box, select **e*Gate API Kit**.
- 5 Follow the on-screen instructions to install the Kit.

Note that the InstallShield program detects and suggests the appropriate **client** installation directory. **Unless you are directed to do so by Oracle support personnel, do not change the suggested installation directory setting.**

- 6 From the **Addons** directory located on the e*Gate installation CD-ROM, navigate to the **ewmux** folder.
- 7 From the **ewmux** folder, select the **eGate_API_win32.taz** file. Expand the file.
- 8 Extract these files to the computer that you have installed the e*Gate API Kit and plan to build your application on.
- 9 If you are running your application from another directory other than where the .dll files reside, you will need to prepend your path to point to the directory that you have extracted your .dlls to, e.g. path c:\mydlls\debug;%path%.

For a list of items installed on the Participating Host machine and committed to the Registry Host, see **“Directories and Files Created by the Installation” on page 45.**

2.2 Installing the e*Gate API Kit on UNIX

2.2.1. 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 non-root user access to begin the e*Gate installation.

2.2.2. Installing the e*Gate API Kit

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

- 1 Log in to 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 and so forth.
- 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.

Caution: *Unless you are directed to do so by Oracle support personnel, do not change the suggested **installation directory** setting.*

- 10 From the **Addons** directory located on the e*Gate installation CD-ROM, navigate to the **ewmux** folder.
- 11 From the **ewmux** folder, select the **eGate_API_xxx.taz** file where xxx is the name of your operating system. Expand the file.
- 12 Extract these files to the computer that you have installed the e*Gate API Kit and plan to build your application on.
- 13 If you are running your application from another directory other than where the .dlls reside, you will need to prepend your path to point to the directory that you have extracted your .dlls to, e.g. path /mydlls/debug;%path%.
- 14 Add execute permissions to the extracted .dlls and .jar files, e.g. **stc_ewimpclnt.dll**.

For a list of items installed on the Participating Host machine and committed to the Registry Host, see **“Directories and Files Created by the Installation”** on page 45.

2.3 Installing the e*Gate API Kit on 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 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 to 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 and so forth.
- 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.

Caution: *Unless you are directed to do so by SeeBeyond support personnel, do not change the suggested **installation directory** setting.*

- 10 From the **Addons** directory located on the e*Gate installation CD-ROM, navigate to the **ewmux** folder.
- 11 From the **ewmux** folder, select the **eGate_API_xxx.taz** file where xxx is the name of your operating system. Expand the file.
- 12 Extract these files to the computer that you have installed the e*Gate API Kit and plan to build your application on.
- 13 If you are running your application from another directory other than where the .dlls reside, you will need to prepend your path to point to the directory that you have extracted your .dlls to, e.g. path /mydlls/debug;%path%.
- 14 Add execute permissions to the extracted .dlls and .jar files, e.g. **stc_ewipmclnt.dll**.

For a list of items installed on the Participating Host machine and committed to the Registry Host, see [“Directories and Files Created by the Installation” on page 45](#).

2.4 Installing the e*Gate API Kit Connecting to an OS/400

Note: *To install the API Kit on your AS/400 system, you must be able to sign on as the security officer.*

2.4.1. Pre-installation

To verify prerequisite software

- 1 From the **Command Entry** screen, enter:

```
go licpgm
```
- 2 From the **Work with Licensed Programs** screen, enter:

10

- 3 In the **Display Installed Licensed Programs** screen, press F11 and verify that *both* of the following are installed:
 - ♦ 5722SS1 at V5R1, *BASE option: Operating System/400.
 - ♦ 5722WDS at V5R1, *BASE option: WebSphere Development ToolSet.
- 4 In the **Display Installed Licensed Programs** screen, press F11 and verify that *at least one* of the following Integrated Language Environment (ILE) programs is installed:
 - ♦ 5722WDS, option 31: Compiler - ILE RPG IV
 - ♦ 5722WDS, option 41: Compiler - ILE COBOL.
 - ♦ 5722WDS, option 51: Compiler - ILE C.
 - ♦ 5722WDS, option 52: Compiler - ILE C++. (Note that this product option requires 5722SS1, option 33: OS/400 - Portable App Solutions Environment.)
- 5 When you have verified the prerequisites, press F3 to exit.

2.4.2. Installing the e*Gate API Kit

To install the e*Gate C API for JMS on an AS/400 system

- 1 If you have not already done so, sign on to the AS/400 system as the security officer.
- 2 Insert the CD-ROM into a drive and enter the following command to load the code and samples, replacing *OPT01* with the name of the drive:

```
rstlib savlib(egateinst) dev(OPT01)
```

- 3 Enter the following command to run the install program:

```
call pgm(egateinst/install)
```

The install program (EGATEINST/INSTALL) should end with the following message:

```
e*Gate API Kit successfully installed
```

- 4 If you do not receive the “successfully installed” message, issue the **dspjoblog** command and check the job log for errors that occurred during installation.
- 5 Take any necessary corrective actions and repeat from step 3.

To verify the installation

- Issue the following commands:

```
dsplib egateapi  
dsplib egatesamp
```

The **Display Library** screen lists the content of each library.

If the libraries exist, and if no error messages were sent during the installation process, then the installation has completed successfully.

For a list of libraries and objects installed, see Table 1.

Table 1 Libraries and Objects Installed to the AS/400 System

Library/Object	Type
EGATEAPI/MSAPI	*SRVPGM (CPPLE)
EGATEAPI/MSCAPI	*SRVPGM (CPPLE)
EGATEAPI/MSCLIENT	*SRVPGM (CPPLE)
EGATEAPI/MSCOMMON	*SRVPGM
EGATEAPI/MSRPGAPI	*SRVPGM (CPPLE)
EGATEAPI/JMSMSGF	*MSGF
EGATESAMP/JMSDEMO	*PGM
EGATESAMP/MAKE	*PGM
EGATESAMP/JMSDEMO	*MODULE
EGATESAMP/JMSUTILS	*MODULE
EGATESAMP/H	*FILE
EGATESAMP/QCLSRC	*FILE
EGATESAMP/QCMDSRC	*FILE
EGATESAMP/QCPPSRC	*FILE
EGATESAMP/QCSRC	*FILE
EGATESAMP/QRPGLESRC	*FILE
EGATESAMP/JMSDEMO	*CMD

2.5 Installing the e*Gate API Kit on an OS/390 or z/OS

2.5.1. Installing from Tape

Submit the following JCL to copy the **STC.RESTORE.JCL** file to disk changing the JCL as required for your installation rules.

```
//JOB Card
//*****
//* COPY e*Gate RESTORE JCL FROM TAPE
//*****
//IEBGENER EXEC PGM=IEBGENER
//SYSPRINT DD SYSOUT=*
//SYSUT1 DD DISP=OLD, DSN=STC.RESTORE.JCL, , VOL=SER=STC390,
// LABEL=(1, SL), UNIT=TAPE
//SYSUT2 DD DISP=(NEW, KEEP),
// DSN=custpfx.STC.RESTORE.JCL,
// UNIT=SYSDA, SPACE=(TRK, (1, 1)),
// DCB=(RECFM=FB, LRECL=80, BLKSIZE=80)
//SYSIN DD DUMMY
```

To copy the load libraries to disk, modify and submit the following JCL (as provided in **STC.RESTORE.JCL**):

```
// JOB CARD
// * CHANGE &TAPEUNIT TO THE NAME OR DEVICE NUMBER OF YOUR TAPE DRIVE
// IEBCOPY EXEC PGM=IEBCOPY
// SYSPRINT DD SYSOUT=*
// SYSIN DD DUMMY
// *****
// * COPY MUX OBJECT FILES TO DISK
// *****
// INOBJ DD DSN=STC.MUX.OBJECT,DISP=OLD,UNIT=&TAPEUNIT,
// VOL=(,RETAIN,SER=STC390),LABEL=(2,SL)
// OUTOBJ DD DSN=&PREFIX..STC.MUX.OBJECT,
// DISP=(NEW,CATLG,DELETE),
// UNIT=SYSDA,
// SPACE=(TRK,(45,15,10),RLSE)
// *****
// * COPY MUX JCL LIBRARY TO DISK
// *****
// INJCL DD DSN=STC.MUX.JCL,DISP=OLD,UNIT=&TAPEUNIT,
// VOL=(,RETAIN,SER=STC390),LABEL=(3,SL)
// OUTJCL DD DSN=&PREFIX..STC.MUX.JCL,
// DISP=(NEW,CATLG,DELETE),
// UNIT=SYSDA,
// SPACE=(TRK,(45,15,10),RLSE)
// SYSIN DD *
COPY INDD=( ( INOBJ,R ) ),OUTDD=OUTOBJ
COPY INDD=( ( INJCL,R ) ),OUTDD=OUTJCL
//
//
```

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

Table 2 Installation Tape Data Sets

Dataset Name	Contents	File Sequence Number
STC.RESTORE.JCL	JCL Sample that is used to unload the remaining two files on the tape.	1
STC.MUX.OBJECT	Compiled object modules.	2
STC.MUX.JCL	Sample JCL to link edit COBOL MUX object modules.	3

2.5.2. Installing from CD

The CD_ROM contains two MVS datasets in `..\setup\addons\ewmux\CobolMUX`:

- **STC.XMIT.MUX.OBJECT**
- **STC.XMIT.MUX.JCL**

Send these datasets, which are in XMIT format, to the mainframe for installation via FTP as follows:

- 1 Allocate MVS datasets to receive the files (for example, **HLQ.XMIT.MUX.OBJECT** and **HLQ.XMIT.MUX.JCL**) with the following:
 - ♦ **RECFM=FB**

- ♦ **LRECL=80**
 - ♦ **BLKSIZE=3120**
 - ♦ **DSORG=PS**
- 2 Send **STC.XMIT.MUX.OBJECT** and **STC.XMIT.MUX.JCL** to the MVS datasets allocated above via FTP, using a binary transfer method. Do not use CRLF or ASCII translation.
 - 3 Issue the TSO Receive command in MVS to restore the files to PDS format:
TSO RECEIVE INDATASET(uploaded.dataset).
 - 4 When prompted by the "INMR906A Enter restore parameters or 'DELETE' or 'END' +" message, enter the following:
DATASET(name.of.your.library) UNIT(unit) VOLUME(volume)

Note: The UNIT() and VOLUME() operands in step 4 above are optional.

2.5.3. Link Editing the COBOL API Object Modules

The shipped object modules must be linked with the version of TCP/IP on the target operating system.

The JCL shown in the samples is located in STC.MUX.JCL and can be modified as necessary to accommodate your system.

Modify the SYSLIB and SYSLMOD datasets to reflect your installation's names for LE, TCP/IP and CICS file names.

Note: If TCP/IP Maintenance is applied to the EZASOKET, the MUX API programs must be relinked.

Sample JCL to Link MUXBAT

```
//LINKMUX JOB ( )
*****
//*                                     *
//* LINK MUXBAT - BATCH COBOL CALLING MUX API KIT *
//*                                     *
//
*****
//STCLINK EXEC PGM=IEWL,REGION=1024K,PARM='MAP,LET,LIST'
//SYSLIB DD DSN=STC.MUX.LOADLIB,DISP=SHR
// DD DSN=TCPIP.SEZATCP,DISP=SHR
// DD DSN=CEE.SCEELKED,DISP=SHR
//SYSLIN DD DSN=STC.MUX.OBJECT(MUXBAT),DISP=SHR
// DD DDNAME=SYSIN
//SYSLMOD DD DSN=STC.MUX.LOADLIB,DISP=SHR
//SYSPRINT DD SYSOUT=*
//SYSUT1 DD UNIT=SYSDA,SPACE=(CYL,(2,2))
//SYSIN DD *
INCLUDE SYSLIB(EZASOKET)
NAME MUXBAT(R)
/*
```

Sample JCL to Link MUXAPI for CICS

```
//MUXLNKC JOB ( ),
```

```

*****
//* LINK MUXAPI - CICS COBOL CALLING MUX API KIT *
//
*****
//LKED EXEC PGM=IEWL,REGION=1024K,PARM='MAP,LET,LIST,RENT'
//SYSLIB DD DSN=CICSTS13.CICS.SDFHLOAD,DISP=SHR
// DD DSN=TCPIP.SEZATCP,DISP=SHR
// DD DSN=CEE.SCEELKED,DISP=SHR
//SYSLIN DD DSN=STC.MUX.OBJECT(MUXAPI),DISP=SHR
// DD DDNAME=SYSIN
//SYSLMOD DD DSN=CICSTS13.CICS.SDFHLOAD,DISP=SHR
//SYSPRINT DD SYSOUT=*
//SYSUT1 DD UNIT=SYSDA,SPACE=(CYL,(2,2))
//SYSIN DD *
INCLUDE SYSLIB(EZACICAL)
NAME MUXAPI(R)
/*

```

Sample to link MUXIMS for IMS

```

//MUXLNKI JOB (),
*****
//* LINK MUXIMS - IMS/TS COBOL CALLING MUX API KIT *
//
*****
//LKED EXEC PGM=IEWL,REGION=1024K,PARM='MAP,LET,LIST'
//SYSLIB DD DSN=STC.MUX.LOADLIB,DISP=SHR
// DD DSN=TCPIP.SEZATCP,DISP=SHR
// DD DSN=CEE.SCEELKED,DISP=SHR
//SYSLIN DD DSN=STC.MUX.OBJECT(MUXIMS),DISP=SHR
// DD DDNAME=SYSIN
//SYSLMOD DD DSN=STC.PROD.MUX.V453.LOADLIB,DISP=SHR
//SYSPRINT DD SYSOUT=*
//SYSUT1 DD UNIT=SYSDA,SPACE=(CYL,(2,2))
//SYSIN DD *
INCLUDE SYSLIB(EZASOKET)
NAME MUXIMS(R)
/*

```

2.5.4. Linking the COBOL API Load Models

To link the COBOL API in the link-editor step of the compile for the calling program, include the following lines:

```

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

```

where MUXxxx is one of the following:

- MUXAPI (for CICS)
- MUXIMS (for IMS)
- MUXBAT (for Batch)

2.5.5. Verifying the CICS Transaction Server Environment for e*Gate

For CICS only: To verify that 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.
```

2.6 Directories and Files Created by the Installation

This section provides the names and locations of files installed during the e*Gate API Kit installation process. It includes information for the following system types:

- **Windows** (page 45)
- **Solaris** (page 47)

2.6.1. Files and Directories - Windows

The e*Gate API Kit installer will install the files listed in [Table 3](#) on your **Windows** system.

Table 3 Files Installed on a Windows System

Location	File
jms\c_api	ms.h
jms\c_api	msc_const.h
jms\c_api	mscapi.h
jms\c_api	msdate.h
jms\c_api	msdll.h
jms\c_api	mslocale.h
jms\c_api	mstypes.h
jms\c_api	msxa.h
jms\c_api	msxid.h
jms\c_api	RTTI.h
jms\c_api	wstring.h
JMS\C_API\Debug	stc_msapi.dll

Table 3 Files Installed on a Windows System

Location	File
JMS\C_API\Debug	stc_msapi.lib
JMS\C_API\Debug	stc_mscapi.dll
JMS\C_API\Debug	stc_mscapi.lib
JMS\C_API\Debug	stc_msclient.dll
JMS\C_API\Debug	stc_msclient.lib
JMS\C_API\Debug	stc_mscommon.dll
JMS\C_API\Debug	stc_mscommon.lib
JMS\C_API\Release	stc_msapi.dll
JMS\C_API\Release	stc_msapi.lib
JMS\C_API\Release	stc_mscapi.dll
JMS\C_API\Release	stc_mscapi.lib
JMS\C_API\Release	stc_msclient.dll
JMS\C_API\Release	stc_msclient.lib
JMS\C_API\Release	stc_mscommon.dll
JMS\C_API\Release	stc_mscommon.lib
jms\com	stc_mscom.dll
JMS\Java	jms.jar
JMS\Java	jta.jar
JMS\Java	stcjms.jar
mux\c	ewipmpclnt.h
mux\c	gendefs.h
mux\c	genererror.h
mux\c	stc_common.dll
mux\c	stc_common.lib
mux\c	stc_ewipmpclnt.dll
mux\c	stc_ewipmpclnt.lib
MUX\Cobol	stc.xmit.mux.cicsload
MUX\Cobol	stc.xmit.mux.load
mux\com	stc_xipmpclnt.dll
mux\com	stcxipmptest.exe
MUX\Java	stcph.jar
MUX\Perl	stc_ewipmpclntperl.dll
MUX\Perl	stc_ewipmpclntperl.pm

2.6.2. Files and Directories - Solaris

The e*Gate API Kit installer will install the files listed in [Table 4](#) on your **Solaris** system.

Table 4 Files Installed on a Solaris System

Location	File
jms\c_api	ms.h
jms\c_api	m_sc_const.h
jms\c_api	m_scapi.h
jms\c_api	m_sdate.h
jms\c_api	m_sdll.h
jms\c_api	m_slocale.h
jms\c_api	m_stypes.h
jms\c_api	m_sxa.h
jms\c_api	m_sxid.h
jms\c_api	RTTI.h
jms\c_api	stc_msapi.dll
jms\c_api	stc_mscapi.dll
jms\c_api	stc_msclient.dll
jms\c_api	stc_mscommon.dll
jms\c_api	wstring.h
JMS\C_API\stlport	algorithm
JMS\C_API\stlport	bitset
JMS\C_API\stlport	cassert
JMS\C_API\stlport	cctype
JMS\C_API\stlport	cerrno
JMS\C_API\stlport	cfloat
JMS\C_API\stlport	climits
JMS\C_API\stlport	locale
JMS\C_API\stlport	cmath
JMS\C_API\stlport	complex
JMS\C_API\stlport	csetjmp
JMS\C_API\stlport	csignal
JMS\C_API\stlport	cstdarg
JMS\C_API\stlport	cstddef
JMS\C_API\stlport	cstdio
JMS\C_API\stlport	cstdlib
JMS\C_API\stlport	cstring

Table 4 Files Installed on a Solaris System

Location	File
JMS\C_API\stlport	cstring.h
JMS\C_API\stlport	ctime
JMS\C_API\stlport	ctype.h
JMS\C_API\stlport	cwchar
JMS\C_API\stlport	cwctype
JMS\C_API\stlport	deque
JMS\C_API\stlport	exception
JMS\C_API\stlport	exception.h
JMS\C_API\stlport	fstream
JMS\C_API\stlport	fstream.h
JMS\C_API\stlport	functional
JMS\C_API\stlport	hash_map
JMS\C_API\stlport	hash_set
JMS\C_API\stlport	iomanip
JMS\C_API\stlport	iomanip.h
JMS\C_API\stlport	ios
JMS\C_API\stlport	ios.h
JMS\C_API\stlport	iosfwd
JMS\C_API\stlport	iostream
JMS\C_API\stlport	iostream.h
JMS\C_API\stlport	istream
JMS\C_API\stlport	istream.h
JMS\C_API\stlport	iterator
JMS\C_API\stlport	limits
JMS\C_API\stlport	list
JMS\C_API\stlport	locale
JMS\C_API\stlport	locale.h
JMS\C_API\stlport	map
JMS\C_API\stlport	math.h
JMS\C_API\stlport	mem.h
JMS\C_API\stlport	memory
JMS\C_API\stlport	memory.new
JMS\C_API\stlport	mmemory.h
JMS\C_API\stlport	new
JMS\C_API\stlport	new.h

Table 4 Files Installed on a Solaris System

Location	File
JMSVC_API\stlport	numeric
JMSVC_API\stlport	ostream
JMSVC_API\stlport	ostream.h
JMSVC_API\stlport	pthread_alloc
JMSVC_API\stlport	queue
JMSVC_API\stlport	rope
JMSVC_API\stlport	set
JMSVC_API\stlport	setjmp.h
JMSVC_API\stlport	signal.h
JMSVC_API\stlport	slist
JMSVC_API\stlport	sstream
JMSVC_API\stlport	stack
JMSVC_API\stlport	stdarg.h
JMSVC_API\stlport	stddef.h
JMSVC_API\stlport	stdexcept
JMSVC_API\stlport	stdio.h
JMSVC_API\stlport	stdlib.h
JMSVC_API\stlport	stl_user_config.h
JMSVC_API\stlport	streambuf
JMSVC_API\stlport	streambuf.h
JMSVC_API\stlport	string
JMSVC_API\stlport	string.h
JMSVC_API\stlport	strstream
JMSVC_API\stlport	strstream.h
JMSVC_API\stlport	time.h
JMSVC_API\stlport	typeinfo
JMSVC_API\stlport	typeinfo.h
JMSVC_API\stlport	utility
JMSVC_API\stlport	valarray
JMSVC_API\stlport	vector
JMSVC_API\stlport	wchar.h
JMSVC_API\stlport	wctype.h
JMSVC_API\stlport\config	_epilog.h
JMSVC_API\stlport\config	_msvc_warnings_off.h
JMSVC_API\stlport\config	_prolog.h

Table 4 Files Installed on a Solaris System

Location	File
JMS\C_API\stlport\config	stl_apcc.h
JMS\C_API\stlport\config	stl_apple.bak.h
JMS\C_API\stlport\config	stl_apple.h
JMS\C_API\stlport\config	stl_as400.h
JMS\C_API\stlport\config	stl_bc.h
JMS\C_API\stlport\config	stl_como.h
JMS\C_API\stlport\config	stl_confix.h
JMS\C_API\stlport\config	stl_dec.h
JMS\C_API\stlport\config	stl_dec_vms.h
JMS\C_API\stlport\config	stl_gcc.h
JMS\C_API\stlport\config	stl_hpacc.h
JMS\C_API\stlport\config	stl_ibm.h
JMS\C_API\stlport\config	stl_intel.h
JMS\C_API\stlport\config	stl_kai.h
JMS\C_API\stlport\config	stl_mlc.h
JMS\C_API\stlport\config	stl_msvc.h
JMS\C_API\stlport\config	stl_mwerks.h
JMS\C_API\stlport\config	stl_mycomp.h
JMS\C_API\stlport\config	stl_sco.h
JMS\C_API\stlport\config	stl_select_lib.h
JMS\C_API\stlport\config	stl_sgi.h
JMS\C_API\stlport\config	stl_sunpro.h
JMS\C_API\stlport\config	stl_symantec.h
JMS\C_API\stlport\config	stl_watcom.h
JMS\C_API\stlport\config	stl_wince.h
JMS\C_API\stlport\config	stlcomp.h
JMS\C_API\stlport\config	vc_select_lib.h
JMS\C_API\stlport\config\new_compiler	confdefs.h
JMS\C_API\stlport\config\new_compiler	config.log
JMS\C_API\stlport\config\new_compiler	configure
JMS\C_API\stlport\config\new_compiler	configure.in
JMS\C_API\stlport\config\new_compiler	readme
JMS\C_API\stlport\config\new_compiler	stlconf.h
JMS\C_API\stlport\config\new_compiler	stlconf.h.in
JMS\C_API\stlport\config\new_compiler	test.cpp

Table 4 Files Installed on a Solaris System

Location	File
JMS\C_API\stlport\config\new_compiler	unconfigure
JMS\C_API\stlport\old_hp	algo.h
JMS\C_API\stlport\old_hp	algbase.h
JMS\C_API\stlport\old_hp	alloc.h
JMS\C_API\stlport\old_hp	bvector.h
JMS\C_API\stlport\old_hp	defalloc.h
JMS\C_API\stlport\old_hp	deque.h
JMS\C_API\stlport\old_hp	function.h
JMS\C_API\stlport\old_hp	hash_map.h
JMS\C_API\stlport\old_hp	hash_set.h
JMS\C_API\stlport\old_hp	hashtable.h
JMS\C_API\stlport\old_hp	heap.h
JMS\C_API\stlport\old_hp	iterator.h
JMS\C_API\stlport\old_hp	list.h
JMS\C_API\stlport\old_hp	map.h
JMS\C_API\stlport\old_hp	multimap.h
JMS\C_API\stlport\old_hp	multiset.h
JMS\C_API\stlport\old_hp	numeric.h
JMS\C_API\stlport\old_hp	pair.h
JMS\C_API\stlport\old_hp	pthread_alloc.h
JMS\C_API\stlport\old_hp	queue.h
JMS\C_API\stlport\old_hp	rope.h
JMS\C_API\stlport\old_hp	set.h
JMS\C_API\stlport\old_hp	slist.h
JMS\C_API\stlport\old_hp	stack.h
JMS\C_API\stlport\old_hp	tempbuf.h
JMS\C_API\stlport\old_hp	tree.h
JMS\C_API\stlport\old_hp	vector.h
JMS\C_API_stlport\stl	_algo.c
JMS\C_API_stlport\stl	_algo.h
JMS\C_API_stlport\stl	_algbase.c
JMS\C_API_stlport\stl	_algbase.h
JMS\C_API_stlport\stl	_alloc.c
JMS\C_API_stlport\stl	_alloc.h
JMS\C_API_stlport\stl	_bitset.c

Table 4 Files Installed on a Solaris System

Location	File
JMS\C_API_stlport\stl	_bitset.h
JMS\C_API_stlport\stl	_bvector.h
JMS\C_API_stlport\stl	_check_config.h
JMS\C_API_stlport\stl	_codecvt.h
JMS\C_API_stlport\stl	_collate.h
JMS\C_API_stlport\stl	_complex.c
JMS\C_API_stlport\stl	_complex.h
JMS\C_API_stlport\stl	_config.h
JMS\C_API_stlport\stl	_construct.h
JMS\C_API_stlport\stl	_ctrails_fns.h
JMS\C_API_stlport\stl	_ctype.h
JMS\C_API_stlport\stl	_deque.c
JMS\C_API_stlport\stl	_deque.h
JMS\C_API_stlport\stl	_epilog.h
JMS\C_API_stlport\stl	_exception.h
JMS\C_API_stlport\stl	_fstream.c
JMS\C_API_stlport\stl	_fstream.h
JMS\C_API_stlport\stl	_function.h
JMS\C_API_stlport\stl	_hash_fun.h
JMS\C_API_stlport\stl	_hash_map.h
JMS\C_API_stlport\stl	_hash_set.h
JMS\C_API_stlport\stl	_hashtable.c
JMS\C_API_stlport\stl	_hashtable.h
JMS\C_API_stlport\stl	_heap.c
JMS\C_API_stlport\stl	_heap.h
JMS\C_API_stlport\stl	_ios.c
JMS\C_API_stlport\stl	_ios.h
JMS\C_API_stlport\stl	_ios_base.h
JMS\C_API_stlport\stl	_iosfwd.h
JMS\C_API_stlport\stl	_istream.c
JMS\C_API_stlport\stl	_istream.h
JMS\C_API_stlport\stl	_iterator.h
JMS\C_API_stlport\stl	_iterator_base.h
JMS\C_API_stlport\stl	_limits.c
JMS\C_API_stlport\stl	_limits.h

Table 4 Files Installed on a Solaris System

Location	File
JMS\C_API_stlport\stl	_list.c
JMS\C_API_stlport\stl	_list.h
JMS\C_API_stlport\stl	_locale.c
JMS\C_API_stlport\stl	_locale.h
JMS\C_API_stlport\stl	_map.h
JMS\C_API_stlport\stl	_messages_facets.h
JMS\C_API_stlport\stl	_monetary.c
JMS\C_API_stlport\stl	_monetary.h
JMS\C_API_stlport\stl	_null_stream.h
JMS\C_API_stlport\stl	_numeric.c
JMS\C_API_stlport\stl	_numeric.h
JMS\C_API_stlport\stl	_numeric_facets.c
JMS\C_API_stlport\stl	_numeric_facets.h
JMS\C_API_stlport\stl	_ostream.c
JMS\C_API_stlport\stl	_ostream.h
JMS\C_API_stlport\stl	_pair.h
JMS\C_API_stlport\stl	_prolog.h
JMS\C_API_stlport\stl	_ptrs_specialize.h
JMS\C_API_stlport\stl	_queue.h
JMS\C_API_stlport\stl	_range_errors.h
JMS\C_API_stlport\stl	_raw_storage_iter.h
JMS\C_API_stlport\stl	_relops.h
JMS\C_API_stlport\stl	_relops_cont.h
JMS\C_API_stlport\stl	_relops_template.h
JMS\C_API_stlport\stl	_rope.c
JMS\C_API_stlport\stl	_rope.h
JMS\C_API_stlport\stl	_set.h
JMS\C_API_stlport\stl	_set_operators.h
JMS\C_API_stlport\stl	_site_config.h
JMS\C_API_stlport\stl	_slist.c
JMS\C_API_stlport\stl	_slist.h
JMS\C_API_stlport\stl	_slist_base.c
JMS\C_API_stlport\stl	_slist_base.h
JMS\C_API_stlport\stl	_sstream.c
JMS\C_API_stlport\stl	_sstream.h

Table 4 Files Installed on a Solaris System

Location	File
JMS\C_API_stlport\stl	_stack.h
JMS\C_API_stlport\stl	_stdio_file.h
JMS\C_API_stlport\stl	_stream_iterator.h
JMS\C_API_stlport\stl	_streambuf.c
JMS\C_API_stlport\stl	_streambuf.h
JMS\C_API_stlport\stl	_string.c
JMS\C_API_stlport\stl	_string.h
JMS\C_API_stlport\stl	_string_fwd.c
JMS\C_API_stlport\stl	_string_fwd.h
JMS\C_API_stlport\stl	_string_hash.h
JMS\C_API_stlport\stl	_string_io.c
JMS\C_API_stlport\stl	_string_io.h
JMS\C_API_stlport\stl	_strstream.h
JMS\C_API_stlport\stl	_tempbuf.c
JMS\C_API_stlport\stl	_tempbuf.h
JMS\C_API_stlport\stl	_threads.c
JMS\C_API_stlport\stl	_threads.h
JMS\C_API_stlport\stl	_time_facets.c
JMS\C_API_stlport\stl	_time_facets.h
JMS\C_API_stlport\stl	_tree.c
JMS\C_API_stlport\stl	_tree.h
JMS\C_API_stlport\stl	_uninitialized.h
JMS\C_API_stlport\stl	_valarray.c
JMS\C_API_stlport\stl	_valarray.h
JMS\C_API_stlport\stl	_vector.c
JMS\C_API_stlport\stl	_vector.h
JMS\C_API_stlport\stl	c_locale.h
JMS\C_API_stlport\stl	char_traits.h
JMS\C_API_stlport\stl	concept_checks.h
JMS\C_API_stlport\stl	container_concepts.h
JMS\C_API_stlport\stl	misl_string.h
JMS\C_API_stlport\stl	stdio_streambuf
JMS\C_API_stlport\stl	type_traits.h
JMS\C_API_stlport\stl\debug	_debug.c
JMS\C_API_stlport\stl\debug	_debug.h

Table 4 Files Installed on a Solaris System

Location	File
JMS\C_API_stlport\stl\debug	_deque.h
JMS\C_API_stlport\stl\debug	_hashtable.h
JMS\C_API_stlport\stl\debug	_iterator.h
JMS\C_API_stlport\stl\debug	_list.h
JMS\C_API_stlport\stl\debug	_slist.h
JMS\C_API_stlport\stl\debug	_string.h
JMS\C_API_stlport\stl\debug	_tree.h
JMS\C_API_stlport\stl\debug	_vector.h
JMS\C_API_stlport\stl\wrappers	_deque.h
JMS\C_API_stlport\stl\wrappers	_hash_map.h
JMS\C_API_stlport\stl\wrappers	_hash_set.h
JMS\C_API_stlport\stl\wrappers	_list.h
JMS\C_API_stlport\stl\wrappers	_map.h
JMS\C_API_stlport\stl\wrappers	_mmap.h
JMS\C_API_stlport\stl\wrappers	_set.h
JMS\C_API_stlport\stl\wrappers	_slish.h
JMS\C_API_stlport\stl\wrappers	_vector.h
JMS\C_API_stlport\using	cstring
JMS\C_API_stlport\using	fstream
JMS\C_API_stlport\using	iomanip
JMS\C_API_stlport\using	ios
JMS\C_API_stlport\using	iosfwd
JMS\C_API_stlport\using	iostream
JMS\C_API_stlport\using	istream
JMS\C_API_stlport\using	locale
JMS\C_API_stlport\using	ostream
JMS\C_API_stlport\using	sstream
JMS\C_API_stlport\using	streambuf
JMS\C_API_stlport\using	stringstream
JMS\C_API_stlport\using\h	fstream.h
JMS\C_API_stlport\using\h	iomanip.h
JMS\C_API_stlport\using\h	iostream.h
JMS\C_API_stlport\using\h	ostream.h
JMS\C_API_stlport\using\h	stringstream.h
JMS\C_API_stlport\wrap_std	complex

Table 4 Files Installed on a Solaris System

Location	File
JMS\C_API_stlport\wrap_std	fstream
JMS\C_API_stlport\wrap_std	iomaniip
JMS\C_API_stlport\wrap_std	ios
JMS\C_API_stlport\wrap_std	iosfwd
JMS\C_API_stlport\wrap_std	iostream
JMS\C_API_stlport\wrap_std	istream
JMS\C_API_stlport\wrap_std	locale
JMS\C_API_stlport\wrap_std	ostream
JMS\C_API_stlport\wrap_std	sstream
JMS\C_API_stlport\wrap_std	streambuf
JMS\C_API_stlport\wrap_std	strstream
JMS\C_API_stlport\wrap_std\h	fstream.h
JMS\C_API_stlport\wrap_std\h	iostream.h
JMS\C_API_stlport\wrap_std\h	streambuf.h
JMS\C_API_stlport\wrap_std\h	strstream.h
JMS\Java	jms.jar
JMS\Java	jta.jar
JMS\Java	stcjms.jar
mux\c	ewipmpclnt.h
mux\c	gendefs.h
mux\c	generror.h
mux\c	stc_common.dll
mux\c	stc_ewipmpclnt.dll
MUX\Cobol	stc.xmit.mux.cicsload
MUX\Cobol	stc.xmit.mux.load
MUX\Java	stcph.jar
MUX\Perl	stc_ewipmpclntperl.dll
MUX\Perl	stc_ewipmpclntperl.pm

2.6.3. Files and Directories - Other Systems

The e*Gate API Kit installer will install the files listed in [Table 5](#) on the following systems:

- AIX
- HP UNIX

- **Linux**

Table 5 Files Installed on AIX, HP UNIX, and Linux Systems

Location	File
jms\c_api	ms.h
jms\c_api	msscconst.h
jms\c_api	mscapi.h
jms\c_api	msdate.h
jms\c_api	msdll.h
jms\c_api	mslocale.h
jms\c_api	mstypes.h
jms\c_api	msxa.h
jms\c_api	msxid.h
jms\c_api	RTTI.h
jms\c_api	wstring.h
JMS\Java	jms.jar
JMS\Java	jta.jar
JMS\Java	stcjms.jar
mux\c	ewipmpclnt.h
mux\c	gendefs.h
mux\c	genererror.h
mux\c	stc_common.dll
mux\c	stc_ewipmpclnt.dll
MUX\Cobol	stc.xmit.mux.cicsload
MUX\Cobol	stc.xmit.mux.load
MUX\Java	stcph.jar
MUX\Perl	stc_ewipmpclntperl.dll
MUX\Perl	stc_ewipmpclntperl.pm

2.6.4. Configuring Perl

If you are using the Multiplexer e*Way, and you wish to use Perl, you will need to configure your PATH environment as follows:

Follow the instructions given in [“Installing the e*Gate API Kit on UNIX” on page 37](#). Once installed, you will need to create a link setup for Perl and set an environment variable as indicated in Table 6.

Table 6 Link Setup

Operating System	Link Command	Library PATH
hpux11	ln -f -s stc_ewimpmpclntperl.dll stc_ewimpmpclntperl.sl	SHLIB_PATH
sparc26	ln -f -s stc_ewimpmpclntperl.dll stc_ewimpmpclntperl.so	LD_LIBRARY_PATH
aix43	ln -f -s stc_ewimpmpclntperl.dll stc_ewimpmpclntperl.so	LIBPATH
linux	ln -f -s stc_ewimpmpclntperl.dll stc_ewimpmpclntperl.so	LD_LIBRARY_PATH

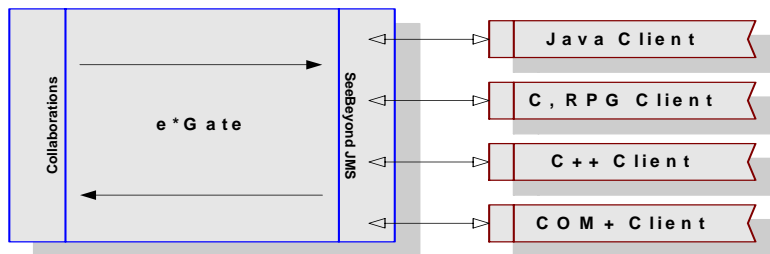
Configuring the Message Service

This chapter explains how to configure the three separate components that constitute the ICAN Suite 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 3 Message Service Communication Architecture

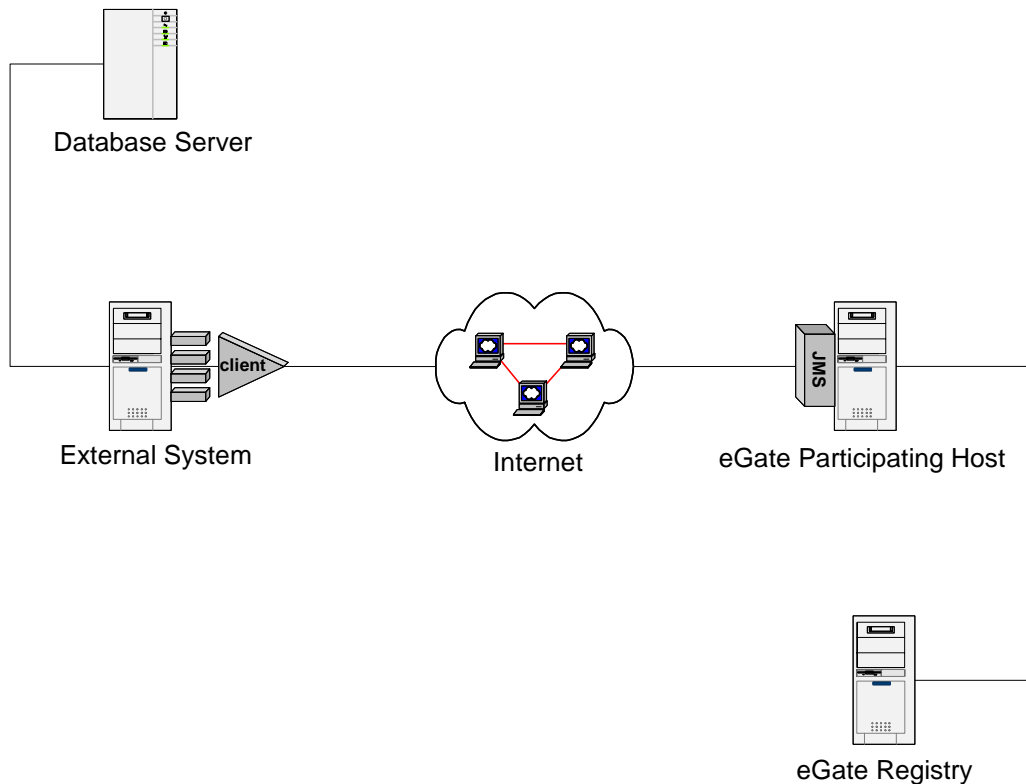


3.1 Configuring the Message Service Clients

The current 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 4 TCP/IP 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 (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 Oracle 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:

Locate the **jms.jar**, **jta.jar**, **stcjms.jar** files in the directory on the external machine that you have installed the e*Gate API Kit on.

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.

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 Oracle 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:

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

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

Viewing the Message Service COM+ APIs

You can view the JMS COM+ APIs using any application that is capable of viewing COM+ APIs. For this illustration on how to view the APIs, we have chosen Microsoft Visual Basic 6.0 as the viewing application.

To begin viewing the APIs

- 1 Start Microsoft Visual Basic 6.0.
- 2 In the **New Project** dialog box, click **Standard EXE** and then click **Open**.
- 3 On the **Project** toolbar, click **References**.
- 4 In the **References** dialog box, select **SeeBeyond Message Service 1.0**, and then click **OK**.
- 5 On the **View** toolbar, click **Object Browser**.
- 6 From the **All Libraries** list box, select **STC_MSCOM**.
- 7 Press the F2 button to open the **Object Browser** dialog box.
- 8 From the **All Libraries** drop-down button, select **STC_MSCOM** to view the supported classes and methods.

- 9 Highlight the class to view the member methods and properties.

Compensating Resource Manager (CRM)

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

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

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

CRM Architecture

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

The functionality provided by the ICAN Suite 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

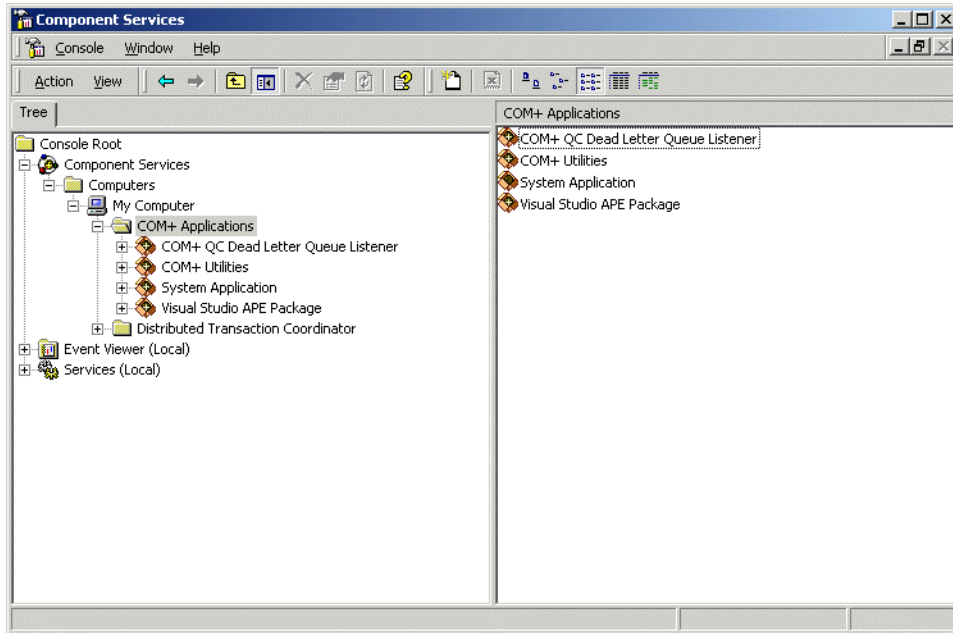
To enable the ICAN Suite CRM functionality, the following steps are required.

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

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

- 2 Open the Component Services applet (**Start -> Settings->Control Panel -> Administrative Tools -> Component Services**).
- 3 Expand the **Component Services** folder (see Figure 5) and right-click **COM+ Applications**.

Figure 5 Component Services Folder



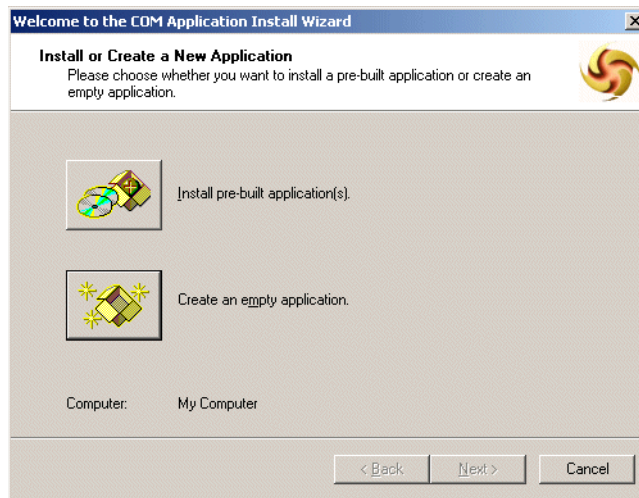
- 4 On the shortcut menu, click **New \ Application**. The COM Application Install Wizard opens (see Figure 6). Click **Next** to continue.

Figure 6 COM Application Install Wizard



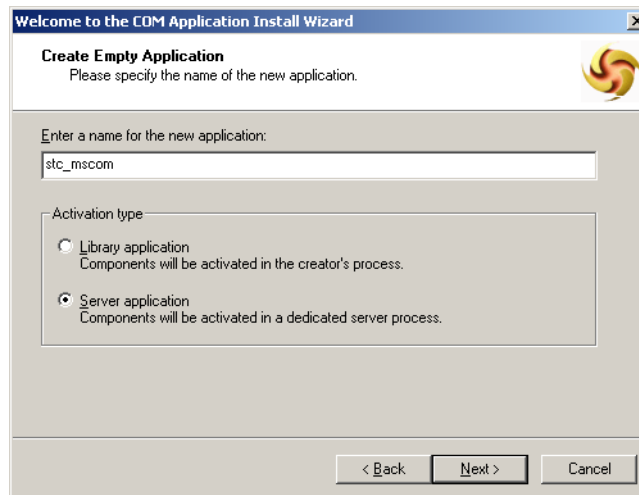
- 5 In the **Install or Create** step (see Figure 7), click **Create an empty application**.

Figure 7 COM Application Install Wizard



- 6 In the **Create Empty Application** step, enter the name `stc_mscom` and click the option button **Server application** (as in Figure 8), and then click **Next**.

Figure 8 COM Application Install Wizard: New Application



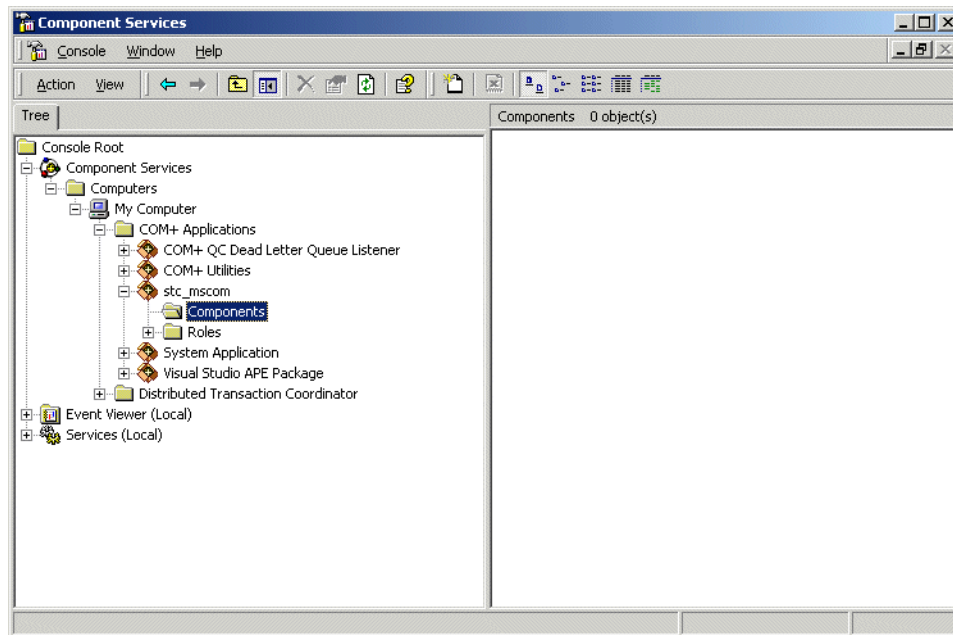
- 7 In the **Set Application Identity** step, click **Interactive User** (as in Figure 9), and then click **Next**.

Figure 9 COM Application Install Wizard: Set Application Identity



- 8 Click **Finish**.
- 9 Expand the **stc_mscom** component; see Figure 10.

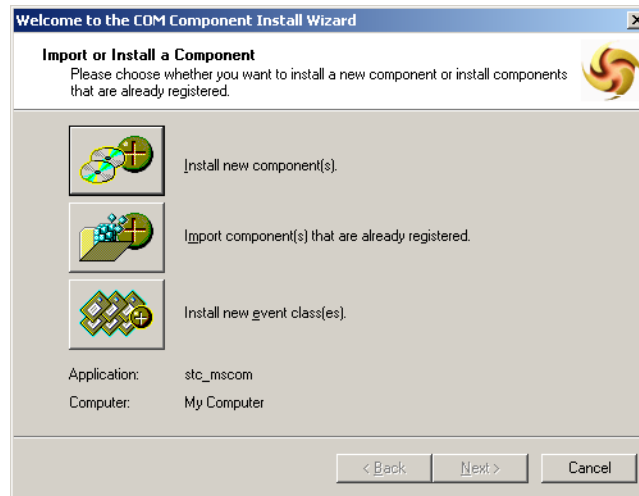
Figure 10 Component Services: stc_mscom Component



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

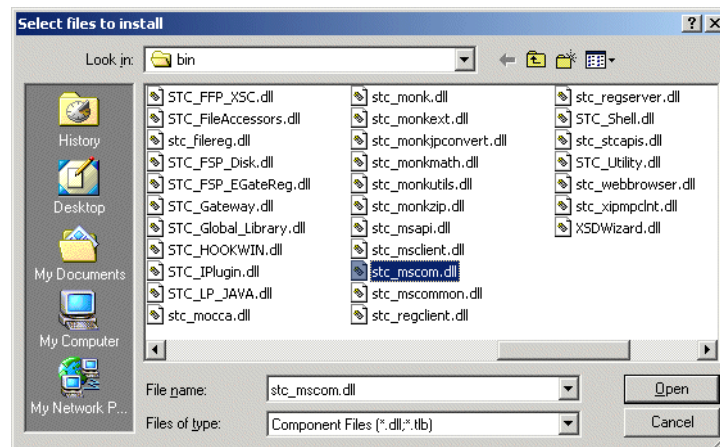
- 11 The COM Component Install Wizard opens (see Figure 11). Click **Install new component(s)**.

Figure 11 COM Component Install Wizard



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

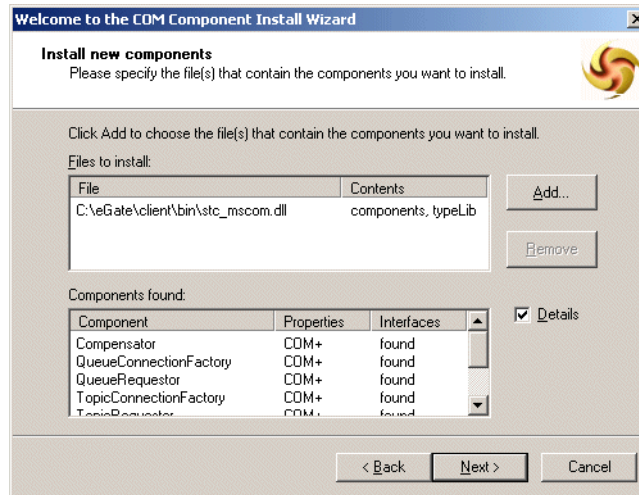
Figure 12 COM Component Install Wizard



- ◆ If you are running the **.dll** on the same machine where e*Gate was installed, the file is located in `<eGate>\client\bin` directory.
- ◆ If **stc_mscom.dll** has been copied to another system, the file is located in the directory where you pasted it previously.

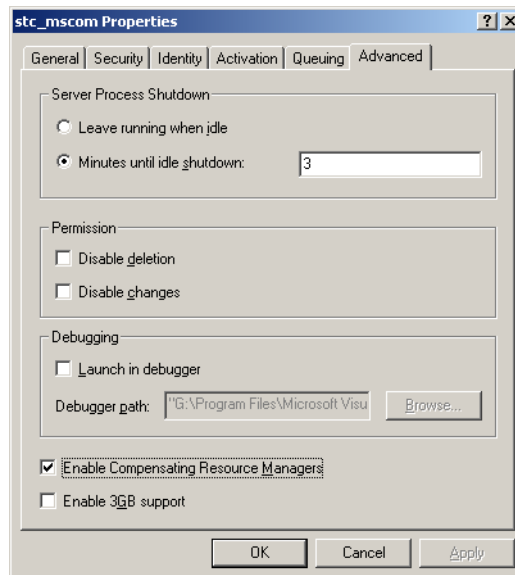
- 13 Once the component appears in the **Components found** pane, ensure that the **Details** box is selected (as in Figure 13), and then click **Next** to continue.

Figure 13 COM Component Install Wizard: Add



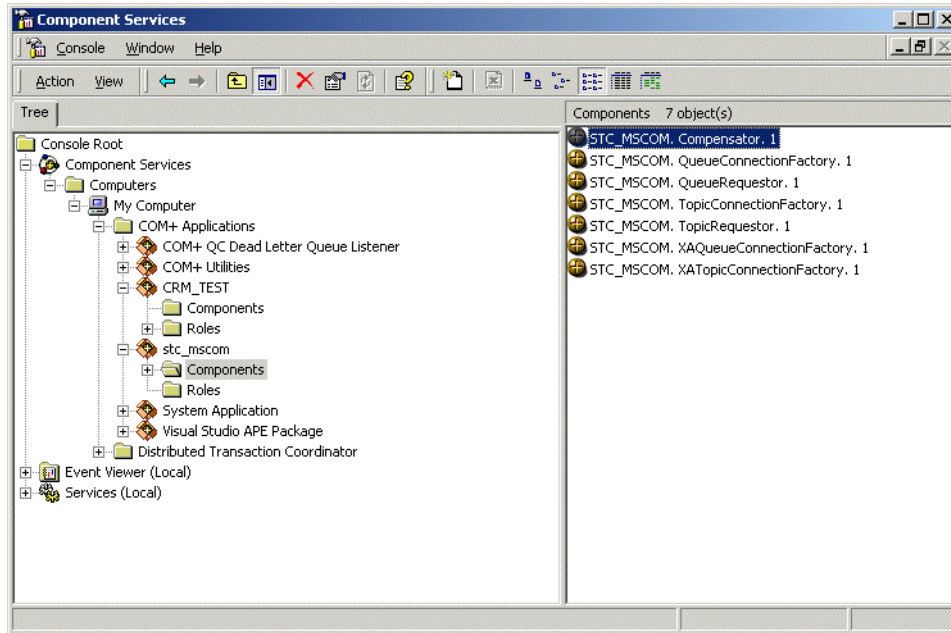
- 14 Click **Finish**.
- 15 Right-click the **stc_mscom** component and, on the shortcut menu, click **Properties**.
- 16 The **stc_mscom Properties** dialog box appears. In the **Advanced** tab, ensure **Enable Compensating Resource Manager** is selected (as in Figure 14), and then click **OK**.

Figure 14 stc_mscom Properties: Advanced



- 17 Expand the **stc_mscom** component and click the **Components** folder to view the objects it contains.
- 18 In the **Components** pane (on the right side of the window; see Figure 15), right-click **STC_MSCOM.Compensator**, and, on the shortcut menu, click **Properties**.

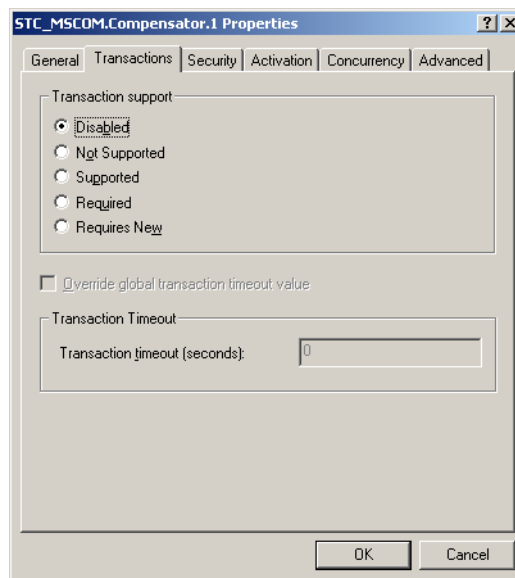
Figure 15 STC_MSCOM.Compensator Properties



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

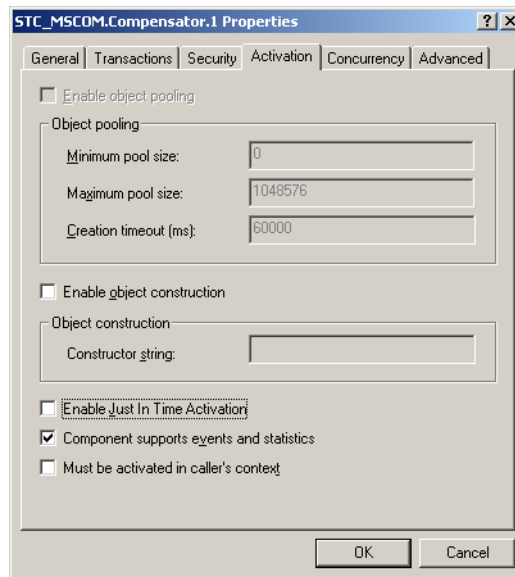
- 19 In the Transactions tab: For **Transaction support**, click **Disabled** (as in Figure 16).

Figure 16 STC_MSCOM.Compensator Properties:Transaction Support



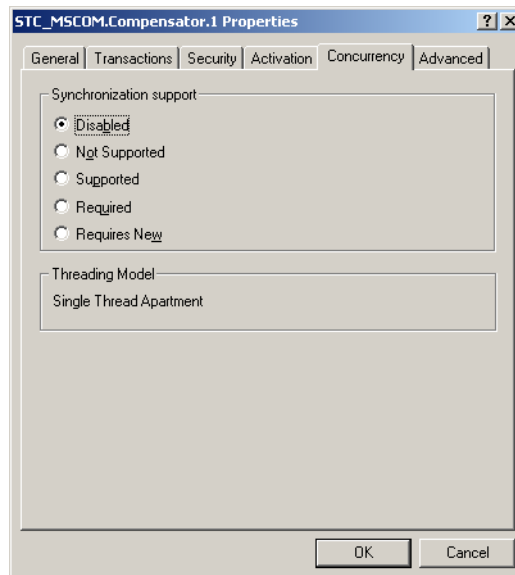
- 20 In the Activation tab: Clear **Enable Just In Time Activation** (as in Figure 17).

Figure 17 STC_MSCOM.Compensator Properties:Activation



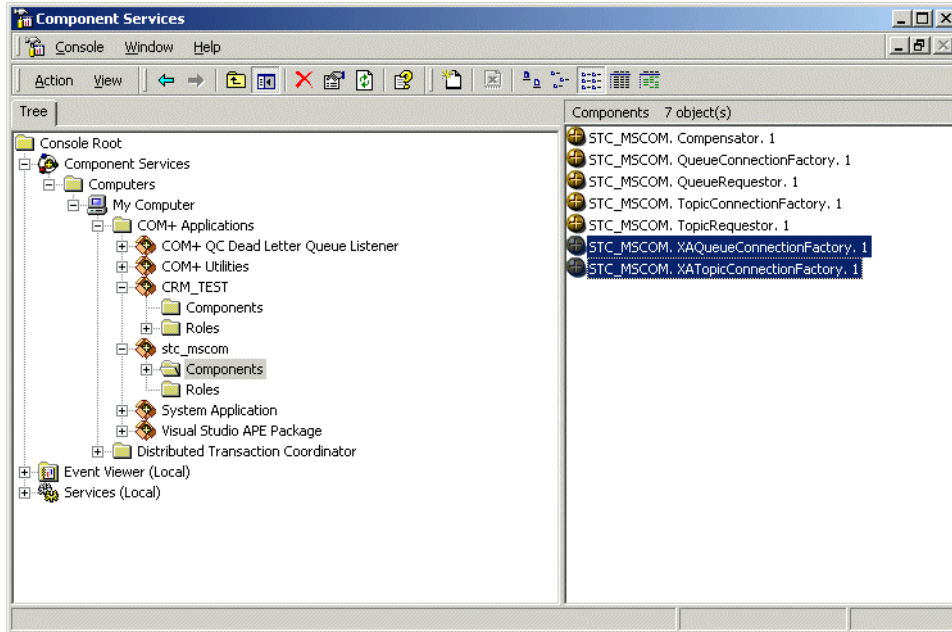
- 21 In the Concurrency tab: For **Synchronization support**, click **Disabled** (as in Figure 18), and then click **OK**.

Figure 18 STC_MSCOM.Compensator Properties:Concurrency



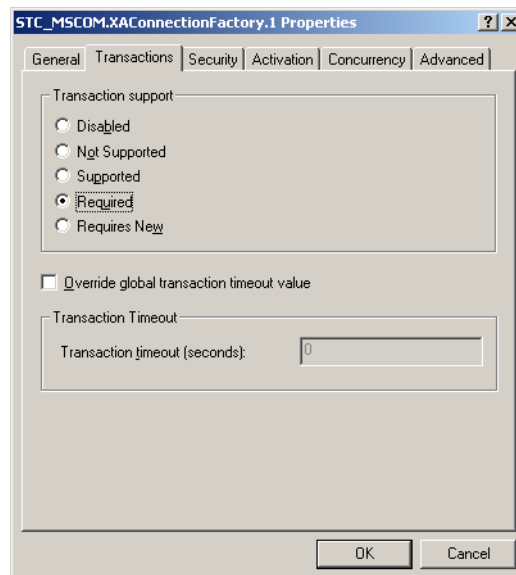
- 22 In the **Components** pane right-click **STC_MSCOM.XAQueueConnectionFactory** and **STC_MSCOM.XATopicConnectionFactory** (as in Figure 19) and, on the shortcut menu, click **Properties**.

Figure 19 STC_MSCOM.XAConnectionFactory Properties



- 23 In the Transactions tab: For **Transaction support**, click **Required** (as in Figure 20).

Figure 20 STC_MSCOM.XAConnectionFactory Properties:Transaction Support



- 24 In the Activation tab: Ensure that **Enable Just In Time Activation** is selected.
- 25 In the Concurrency tab: For **Synchronization support**, click **Required**, and then click **OK**.

3.2 Configuring the Message Server

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

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

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

Implementing JMS

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

To implement JMS in Java or COM+

- For information on what you need to use the APIs, see [“Implementing JMS Models in Java or COM+” on page 73](#).
- For sample code, see [“Sample Code for Using JMS in Java and COM+” on page 76](#).
- For a discussion of the sample schema, see [“Sample Schema Implementation” on page 119](#).

To implement JMS in C++

- For information on what you need to use the APIs, see [“Sample Code for Using JMS in C++” on page 175](#).
- For sample code, see [“Sample Code for Using JMS in C++” on page 175](#).
- For a discussion of the sample schema, see [“Sample Schema Implementation” on page 119](#).

4.1 Implementing JMS Models in Java or COM+

This section discusses how to use the JMS Java 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 JMS API” on page 197](#).

- 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 used by both clients and providers to identify and route messages.
- **Properties** - Properties provide a way to add optional header fields to messages
 - ♦ Application-specific
 - ♦ Standard properties
 - ♦ Provider-specific
- **Body (or Payload)** - JMS provides for supporting different types of payload. The current JMS 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.

JMSDeliveryMode

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

JMSMessageID

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

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

JMSTimestamp

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

JMSExpiration

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

JMSRedelivered

The JMSRedelivered header field 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 (Payload)

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

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

4.2 Sample Code for Using JMS in Java and COM+

Code samples are provided on the product CD-ROM as shown in Table 7.

Table 7 Sample Code for Using JMS in Java and COM+

Location	Contents
samples\jmsapi	File jmsdemo.zip (sample schema): See "Sample Schema Implementation" on page 119 .
samples\jmsapi\java	Java source code samples: <ul style="list-style-type: none"> ▪ Publisher.java: See "Java Publish" on page 78. ▪ Subscriber.java: See "Java Subscribe" on page 79. ▪ Sender.java: See "Java Point-to-Point Sender" on page 84. ▪ Receiver.java: See "Java Point-to-Point Receiver" on page 85. ▪ SampleTopicRequestor.java: See "Java TopicRequestor" on page 90. ▪ SampleQueueRequestor.java: See "Java QueueRequestor" on page 91. ▪ SelectorPublisher.java: See "Java Message Selector Publisher" on page 102. ▪ SelectorSubscriber.java: See "Java Message Selector Subscriber" on page 104. ▪ XAPublisher.java: See "Java XA Publisher" on page 106. ▪ XASubscriber.java: See "Java XA Subscriber" on page 108.

Table 7 Sample Code for Using JMS in Java and COM+

Location	Contents
samples\jmsapi\com	<p>Samples for COM+:</p> <ul style="list-style-type: none"> ▪ CRM_Sample: See “The Compensating Resource Manager” on page 112. ▪ MessageSelector_Sample: See “COM VB Message Selector” on page 105. ▪ Point2Point_Sample: See “COM VB Point-to-Point” on page 87. ▪ PublishSubscribe_Sample: See “COM VB Publish/Subscribe” on page 81. ▪ QueueRequestor_Sample: See “COM VB QueueRequestor” on page 94. ▪ TopicRequestor_Sample: See “COM VB TopicRequestor” on page 94. ▪ XA_Sample: See “COM VB XA Sample” on page 111.

The external source 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.

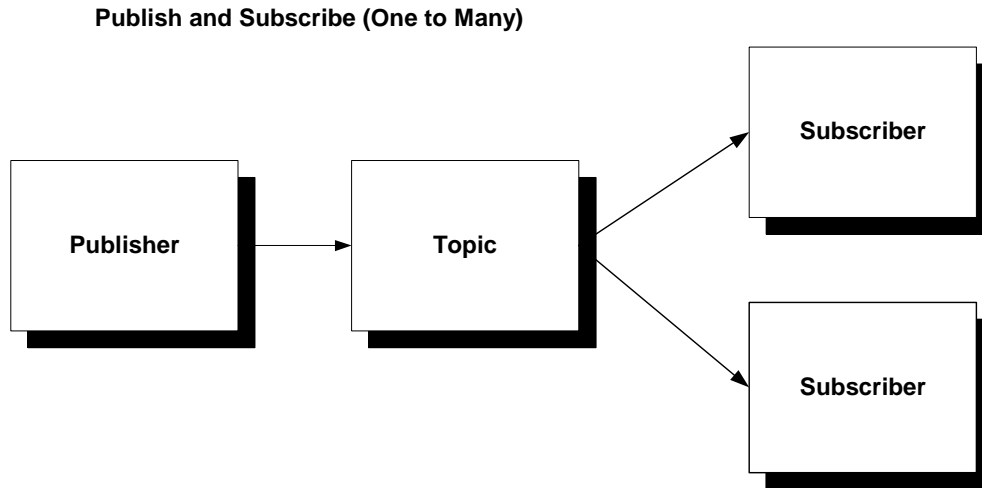
4.2.1. The Publish/Subscribe Model

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 are pushed via a channel referred to as a topic. This topic is considered a destination to which producers publish and consumers subscribe. Messages are automatically pushed to all qualified consumers.
- There is no coupling of the producers to the consumers. Both subscribers and publishers can be dynamically added at runtime, allowing the system to change as needed.
- Each client receives a copy of the messages that have been published to those topics to which it subscribes. Multiple subscribers can receive messages published by one producer.

Figure 21 below illustrates a basic Publish/Subscribe schema.

Figure 21 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 bytesmessage or textmessage.
- 4 Send messages, varying the bytes or text if desired.
- 5 When all messages have been sent, close the connection.

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

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


- 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.
             */
        }
    }
}
```



```
' 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

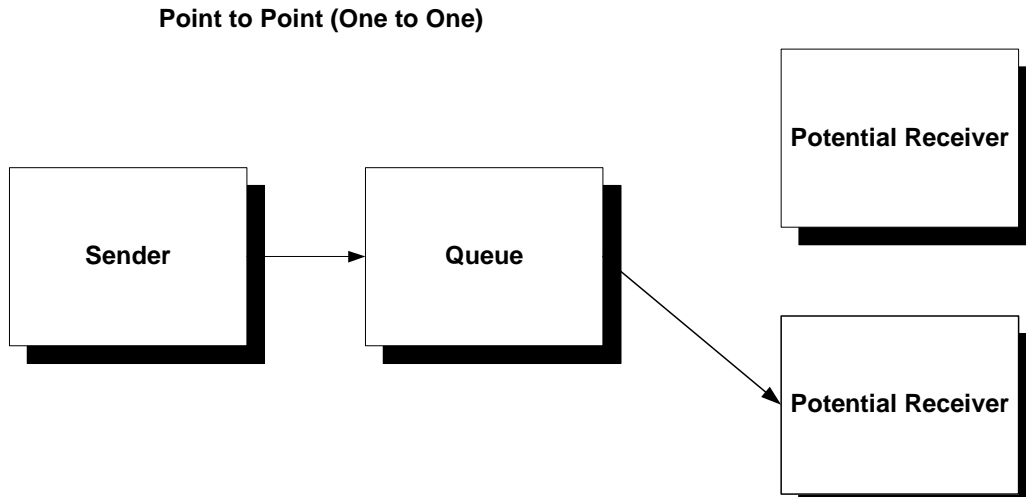
Point-to-Point messaging is based on the sending of a message to a named destination (as is the publish/subscribe model). There is no direct coupling of the producers to the consumers. One main difference between point-to-point and publish/subscribe messaging is that in the first, messages are delivered, 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 from the "front of the line".

- Receivers and senders can be added dynamically at runtime, allowing the system to grow as needed.

Figure 22 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("-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) {
            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: " +

```



```

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

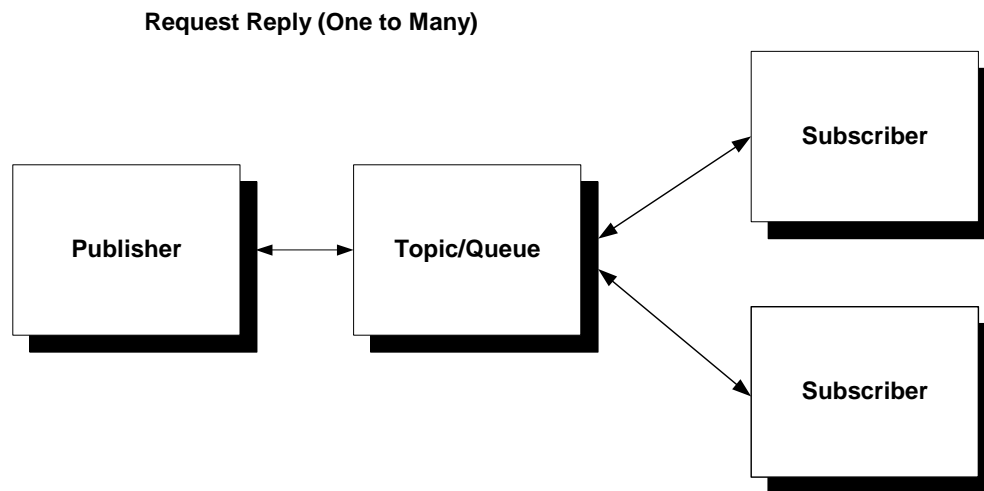
```


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 21 below illustrates a basic Request-Reply schema.

Figure 23 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 **Oracle 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 requeust 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\n" +
            "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(JMSEException je)
{
    je.printStackTrace();
}
}

public void onException(JMSEException 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 function create QueueRequestor and reply Subscriber.
     * Send request 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(JMSEException je)
{
    je.printStackTrace();
}
}

public void onException(JMSEException 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.CreateTopicSession, 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 NamingEvents.
 - ♦ 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 v1.6.0_19, 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) v1.6.0_19 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.

If you are not using JNDI as an installed extension, 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 `HashMap`) 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 `InitialContext` constructor.

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.naming.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** typeface.

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

```
C:\temp>set CLASSPATH=
%CLASSPATH%;
C:\jndi\fscontext1_2beta3\lib\fscontext.jar;
C:\jndi\fscontext1_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("APTTopic", (Topic) top);
}
catch (javax.naming.NameAlreadyBoundException e) {

}

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

} catch( Exception e ) {
```

```
        e.printStackTrace();
    }
}
```

At this point the retrieved objects can now communicate with the 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 syntax is composed, according to a subset of the SQL92* conditional expression syntax. If the value of a message selector is provided as an empty string, the value is treated as “null” and indicates that there is no message selector for the message consumer.

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

Identifiers

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

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

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

The following JMS headers can be used as identifiers:

- JMSDeliveryMode
- JMSPriority
- JMSMessageID
- JMSTimestamp

- JMSCorrelationID
- JMSType

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

- JMSDestination
- JMSReplyTo

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

Literals

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

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

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

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

Declaring a Message Selector

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

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

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

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

The message selector (“username <> 'John'”) tells the message 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(selector) 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);

```



```

    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.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("JMSEException in onMessage(): "
            + e.toString());
    }
}
```


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
```

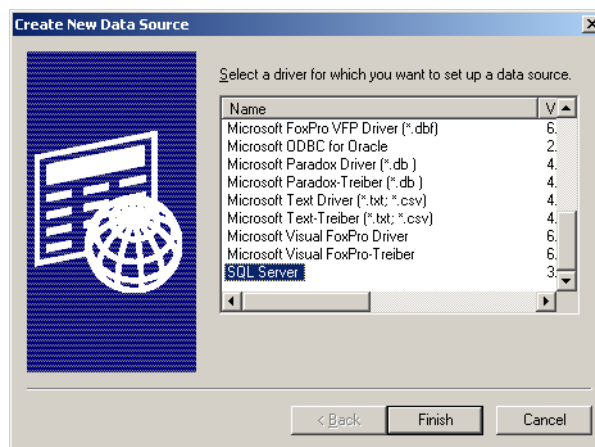
4.2.7. The Compensating Resource Manager

Creating a SQL Server Database

The samples provided are designed using an SQL Server Database.

- 1 Create a SQL Server database, using the name “CRM” for the purpose of testing the samples.
- 2 Create a table, using the name “Messages”.
- 3 Create two columns in the table, “UID” and “Message”.
- 4 From Settings->ControlPanel->AdministrativeTools->>DataSources, Add an SQL Database source.

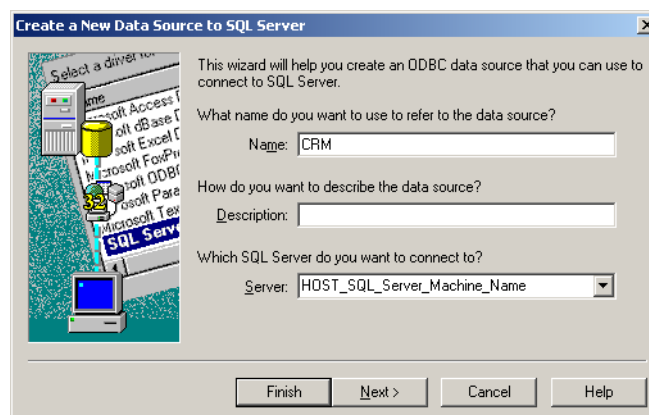
Figure 24 SQL Database Source



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

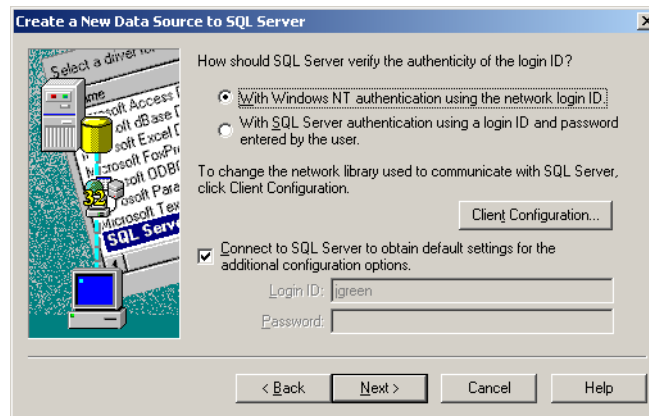
Important: You will not be able to continue until a successful connection is made.

Figure 25 SQL Datasource



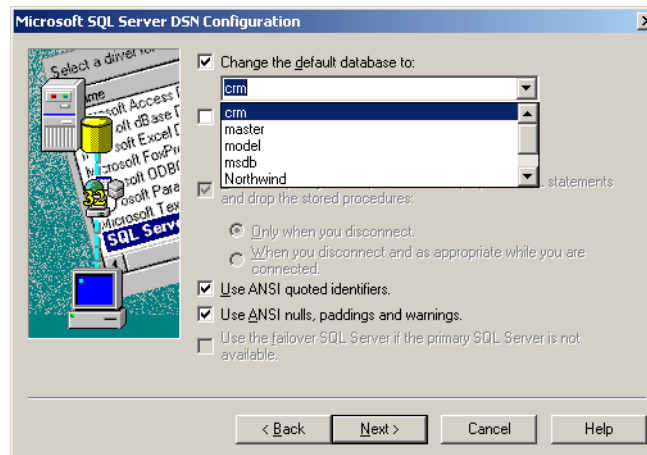
- 6 Ensure that the authentication and login settings correspond to Figure 26 below.

Figure 26 Login Settings



- 7 Select the recently created database as the default from the drop-down list, as shown in Figure 27 below. Click Next to continue.

Figure 27 Default SQL Server Database



- 8 Click Finish.

Configuring the Compensating Resource Manager (CRM)

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

Both the CRM Worker and Compensator are COM+ components and they must be configured using the Windows 2000 Component Services Explorer function properly. The CRM Worker and CRM Compensator must be installed into the same COM+ application that was completed above. (See [“Configuring the Compensating Resource Manager \(CRM\)” on page 113](#) for more information.)

Note: You must create the "CRM" database before attempting to use any sample code. See ["Creating a SQL Server Database" on page 112](#) 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

CRMTTest.vdb

CRMTTest.dll

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

InsertMessage.cls

Option Explicit

```
Sub Add(message As String, commit As Boolean)
    On Error GoTo errorHandler
    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
errorHandler:
    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 errorHandler
    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
errorHandler:
    ObjCtx.SetAbort
    Err.Raise Err
End Sub

Sub Receive(topicName As String, msg As String, commit As Boolean)
    On Error GoTo errorHandler
    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
errorHandler:
    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 **CRMTTest.dll** by performing the following:

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

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

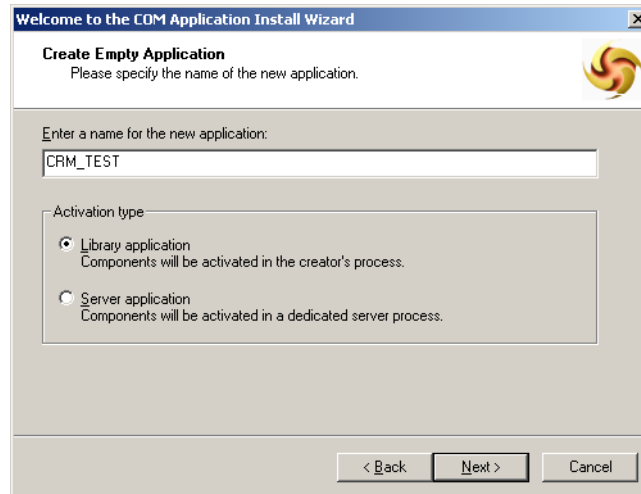
- 4 From the following location:

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 28 CRM_TEST Application



- 6 Click Next to continue, and then click Finish.
- 7 Expand the CRM_TEST component, right-click the Components folder, and then click New Component. The COM Component Install Wizard opens. Click Next to continue.
- 8 Click Install New Component(s). Browse to the location of the recently compiled CRMTest.dll. Click Open. Accept the remainder of the default settings.

4.3 Sample Schema Implementation

The sample schema implementation is available in the **samples\jmsapi** directory of the product media. You can import this schema to Schema Designer either when you initially log in, or when you are already running an active session.

To import the schema upon logging in

- 1 Enter values for Server, Login ID, and Password, and then click **Log In**.
- 2 In the **Open Schema on Registry Host** dialog box, click **New**.
- 3 In the **New Schema** dialog box, select **Create from export**, and then click **Find**.
- 4 In the **Import from File** dialog box, navigate to **samples\jmsapi** directory, select **jmsdemo.zip**, click **Open**, enter your name for the schema, and then click **Open**.

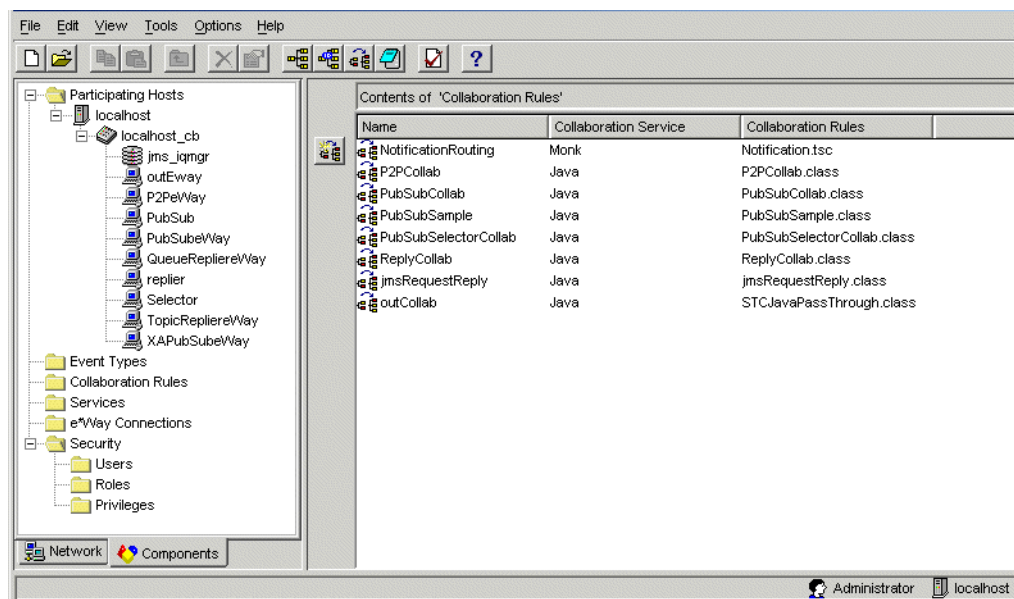
To import the sample schema during an active session

- 1 From the Schema Designer, on the **File** menu, click **New Schema** (Ctrl+N).
- 2 In the **New Schema** dialog box, select **Create from export**, and then click **Find**.
- 3 In the **Import from File** dialog box, navigate to **samples\jmsapi** directory, select **jmsdemo.zip**, click **Open**, enter your name for the schema, and then click **Open**.

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 29 JMS Sample Schema, Schema Designer View



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

- Oracle SeeBeyond JMS IQ Manager
- Event Types
- Event Type Definition
- JMS e*Way Connections
- Java Collaboration Rules
- Multi-Mode e*Ways
- File e*Ways
- Java Collaborations

Oracle SeeBeyond JMS IQ Manager

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

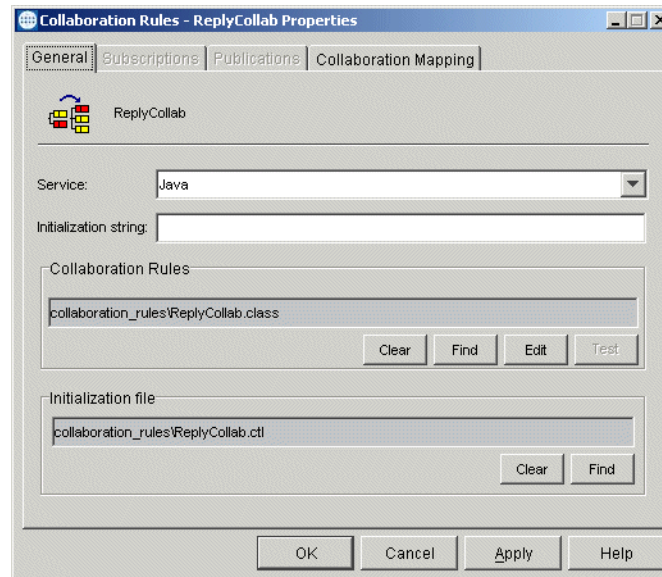
Java Collaboration Rules

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 30 jmsRequestReply



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:

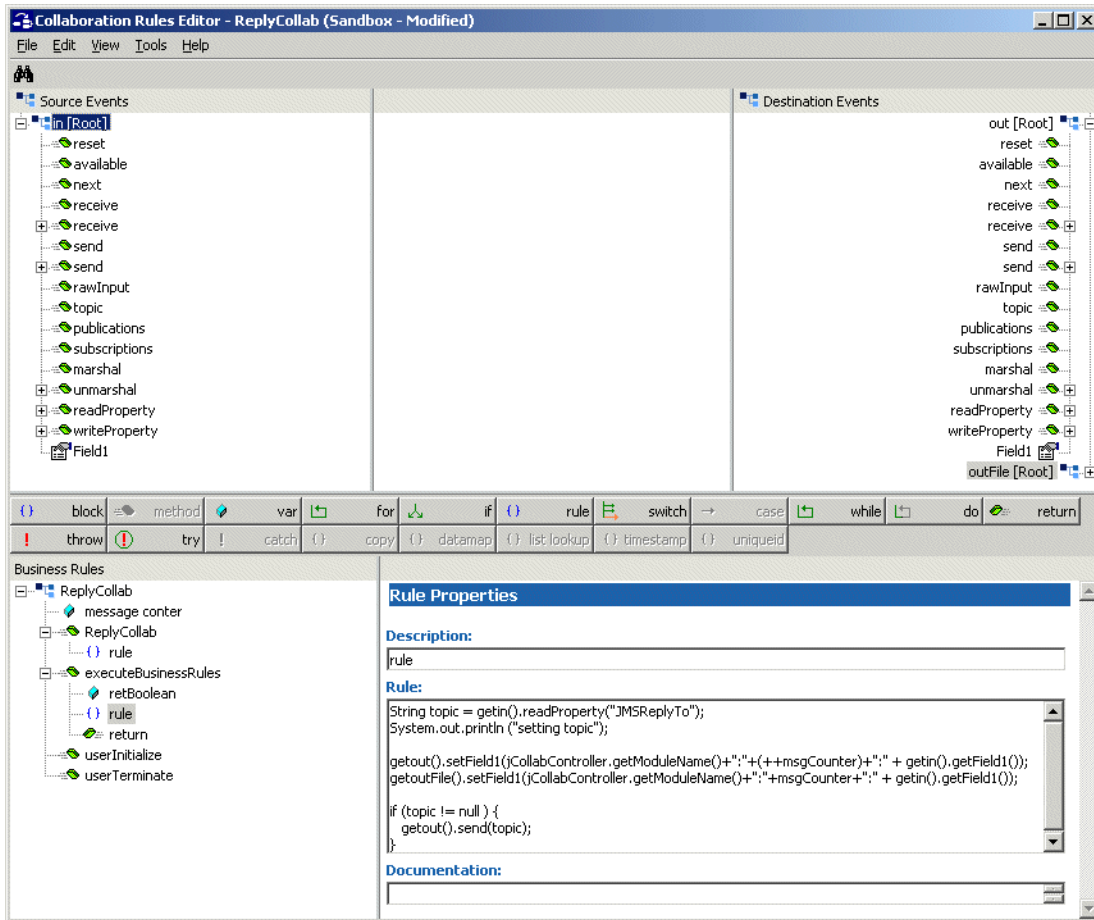
```
getout().setField1(jCollabController.getModuleName()+":"+ (++msgCounter)+": "+ getin().getField1());

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

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

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

Figure 31 ReplyCollab Rule



Compile, Save and Promote to run time.

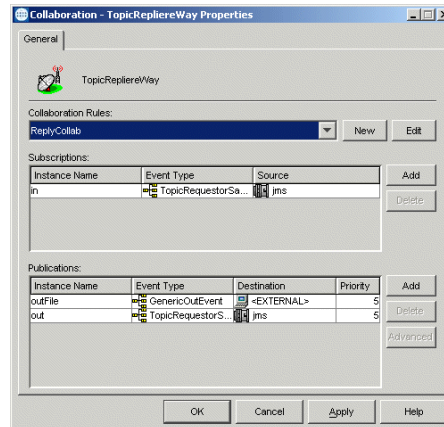
Multi-Mode e*Way

The Multi-Mode e*Way (replier and outEWay), along with the e*Way Connection, provides connectivity to the external system via the Oracle 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

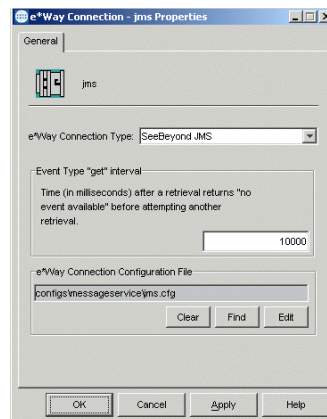
Associated with the e*Way, the Collaboration designates the functionality as defined in the Collaboration Rule with the e*Way. The Collaboration displays the Event Type and Source for both the Subscription(s) and Publication(s). It is very important that these values be set according to the expected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

Figure 32 TopicReplierWay Collaboration



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 33 e*Way Connection



4.3.2. Executing the Schema

From the command line, start the Control Broker by entering the following:

```
stccb.exe -rh <host> -rs <schema_name> -un Administrator -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 point to the Participating Host on which the JMS IQ Manager is running.

4.4 Implementing JMS Models in C

This section discusses how to use the JMS C APIs and their wrappers in C to exchange data with an e*Gate system.

4.4.1. Wrapper Functions for C

For C, the API Kit supplies a set of JMS wrapper functions. The purpose of this layer is to help developers focus on the programming task rather than on the details of JMS. Thus, while the wrapper functions are sufficient for most applications, they do not provide a complete function set; for details on the complete C API, see [“The C API for JMS” on page 355](#).

At this higher level of abstraction, you need only manage a few types of structures:

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

For each of these structures, the wrapper provides the equivalent of a constructor (`CrtObj` or `OpnObj`) and a destructor (`CloseObj` or `DltObj`). The other wrapper functions allow you to write/read/send/receive messages, to set up request/reply messaging, and to commit a transacted session.

The wrapper functions use parameters `err` and `errBuf` to handle error codes and error message text; see [“Differences Between Java and C in Error Handling” on page 456](#).

4.5 Sample Code for Using JMS in C

From the **Samples** directory located on the e*Gate installation CD-ROM, navigate to the **jmsapi** folder. Select the **C** folder and extract the sample files from this folder to the

computer that you have installed the e*Gate API Kit on. The samples were built using Microsoft Visual Studio 6.0.

For the JMS API in C, the following sample programs are provided on the product CD-ROM:

- [Publish/Subscribe Messaging Using C](#) on page 126
- [Queue Messaging \(Sending/Receiving\) Using C](#) on page 131
- [Request-Reply Messaging Using C](#) on page 136
- [Message Selector Using C](#) on page 140
- [“Publish/Subscribe Messaging Using XA in C”](#) on page 145

Note: For multi byte data processing on non-English operating systems, use the following method to set your locale leaving the double quote blank. This will allow the program to pick up your system's default locale setting.

```
setlocale(LC_CTYPE, "");
```

4.5.1. Publish/Subscribe Messaging Using C

```
(1)  *-----*
(2)  * Sample code to demonstrate JMS Pub/Sub Messaging.
(3)  *-----*
(4)  *
(5)  * Disclaimer:
(6)  *
(7)  * Copyright 2002 by SeeBeyond Technology Corporation.
(8)  * All Rights Reserved.
(9)  * Unless otherwise stipulated in a written agreement for this
(10) * software, duly executed by SeeBeyond Technology Corporation, this
(11) * software is provided as is without warranty of any kind. The
(12) * entire risk as to the results and performance of this software
(13) * is assumed by the user. SeeBeyond Technology Corporation disclaims
(14) * all warranties, either express or implied, including but not
(15) * limited, the implied warranties of merchantability, fitness for a
(16) * particular purpose, title and non-infringement, with respect to
(17) * this software.
(18) *
(19) *-----*/
(20) #include "mscapi.h"
(21) #include <stdlib.h>
(22) #include <stdio.h>
(23) #include <string.h>
(24) #ifndef WIN32
(25) #include <unistd.h>
(26) #endif
(27) #if defined(WIN32)
(28) #include "sbyn_getopt.h"
(29) #endif
(30)
(31) #if defined(OS400)
(32) extern char *optarg;
(33) #endif
(34)
(35) #if defined(__gnu__)
(36) #include <getopt.h>
(37) #endif
(38)
```

```
(39) char          optionProducer[] = "[ -u ] [ -p port ]
(40)              [ -h hostname ]";
(41) char          optionConsumer[] = "[ -c ] [ -p port ]
(42)              [ -h hostname ]";
(43) char          optdescription[] = "\t-u    run as a
(44)              producer\n\t-c run as a consumer\n\t-p
(45)              port number\n\t-h    hostname\n";
(46) static char    localhost[] = "localhost";
(47) static unsigned short susPort = 24053; /* default port number */
(48) unsigned long   sulMessageSize = 16; /* default host name */
(49) static char*    spHostName;
(50) static void     subscriber();
(51) static void     publisher();
(52)
(53) /* Check for errors. */
(54) static void check_error(int err, char* errBuf, int exitnow)
(55) {
(56)     if (err){
(57)         printf("ERROR:0x%x - %s\n", err, errBuf);
(58)         if (exitnow)
(59)             exit(1);
(60)     }
(61) }
(62)
(63) int main(int argc, char *argv[]) {
(64)     int          c;
(65)     char          cOption = 0;
(66)
(67)     spHostName = localhost;
(68)
(69)     while((c = getopt(argc, argv, ":p:h:P:H:cCUu")) != -1) {
(70)         switch(c){
(71)             case 'p':
(72)             case 'P':
(73)                 susPort = atoi(optarg); /* setup the port number */
(74)                 break;
(75)             case 'h':
(76)             case 'H':
(77)                 spHostName = optarg; /* setup the hostname */
(78)                 break;
(79)             case 'U':
(80)             case 'u':
(81)                 cOption = 'u'; /* run as a producer */
(82)                 break;
(83)             case 'c':
(84)             case 'C':
(85)                 cOption = 'c'; /* run as a consumer */
(86)                 break;
(87)             case ':':
(88)             case '?':
(89)                 printf("\nSYNOPSIS\n");
(90)                 printf("%s %s\n", argv[0], optionProducer);
(91)                 printf("%s %s\n", argv[0], optionConsumer);
(92)                 printf("%s\n", optdescription);
(93)                 exit(1);
(94)                 break;
(95)         }
(96)     }
(97)
(98)     if (cOption == 'u'){
(99)         publisher(); /* invoke producer */
(100)    } else if (cOption == 'c'){
(101)        subscriber(); /* invoke consumer */
(102)    } else {
```

```
(103)         printf("\nSYNOPSIS\n");
(104)         printf("%s %s\n", argv[0], optionProducer);
(105)         printf("%s %s\n", argv[0], optionConsumer);
(106)         printf("%s\n", optdescription);
(107)         exit(1);
(108)     }
(109) }
(110)
(111)
(112) /*
(113) * =====
(114) * Topic Publisher
(115) * This routine demonstrates how to publish to a topic.
(116) * =====
(117) */
(118) static void publisher() {
(119)     SBYN_TopicPublisher*   pTopicPublisher;
(120)     SBYN_Session*          pTopicSession;
(121)     SBYN_Destination*      pTopic;
(122)     SBYN_Message*          pTextMessage;
(123)     SBYN_Connection*       pTopicConnection;
(124)     SBYN_TopicConnectionFactory* pTcf;
(125)     int                     iErr;
(126)     char                     szErrBuf[256];
(127)     char                     pBuffer[] = "This is a text message";
(128)     char                     pTopicName[] = "PubSubSample";
(129)     int                     iMessagePriority = 4;
(130)     long                    iTimeToLive = 0;
(131)
(132)     /* Create a topic factory. */
(133)     pTcf = CreateTopicConnectionFactory(spHostName, susPort, &iErr,
(134)     szErrBuf);
(135)     check_error(iErr, szErrBuf, 1);
(136)     if(!pTcf) {
(137)         printf("CreateTopicConnectionFactory
(138) failed\nError:%0X\nReason:%s\n", iErr, szErrBuf);
(139)         exit(2);
(140)     }
(141)
(142)     /* Create a topic connection. */
(143)     pTopicConnection = CreateTopicConnection(pTcf,&iErr, szErrBuf);
(144)     check_error(iErr, szErrBuf, 1);
(145)
(146)     /* Set the client ID. */
(147)     ConnectionSetClientID(pTopicConnection,
(148)     (char*)"TopicTestPublisher", &iErr, szErrBuf);
(149)     check_error(iErr, szErrBuf, 1);
(150)
(151)     /* Start the connection. */
(152)     ConnectionStart(pTopicConnection,&iErr, szErrBuf);
(153)     check_error(iErr, szErrBuf, 1);
(154)
(155)     /* Create a topic session. */
(156)     pTopicSession = ConnectionCreateTopicSession (pTopicConnection,
(157)     SBYN_TRANSACTED, SBYN_CLIENT_ACKNOWLEDGE, &iErr, szErrBuf);
(158)     check_error(iErr, szErrBuf, 1);
(159)     if(!pTopicSession) {
(160)         printf("CreateTopicSession failed\nError:%0X\nReason:%s\n",
(161)     iErr, szErrBuf);
(162)         exit(2);
(163)     }
(164)
(165)     /* Create a topic. */
(166)     pTopic = SessionCreateTopic(pTopicSession, pTopicName, &iErr,
```



```
(167)     szErrBuf);
(168)     check_error(iErr, szErrBuf, 1);
(169)
(170)     /* Create a topic publisher. */
(171)     pTopicPublisher = SessionCreatePublisher(pTopicSession, pTopic,
(172)     &iErr, szErrBuf);
(173)     check_error(iErr, szErrBuf, 1);
(174)
(175)     /* Create a text message. */
(176)     pTextMessage = SessionCreateTextMessage(pTopicSession, &iErr,
(177)     szErrBuf);
(178)     check_error(iErr, szErrBuf, 1);
(179)
(180)     /* Clear the body (payload) of the message. */
(181)     ClearBody(pTextMessage, &iErr, szErrBuf); /* set the mode to r/w */
(182)     check_error(iErr, szErrBuf, 1);
(183)
(184)     /* Copy in the text to be sent. */
(185)     SetText(pTextMessage, pBuffer, &iErr, szErrBuf);
(186)     check_error(iErr, szErrBuf, 1);
(187)
(188)     /* Set the JMSType of the message to "ASCII". */
(189)     SetJMSType(pTextMessage, (char*)"ASCII", &iErr, szErrBuf);
(190)     check_error(iErr, szErrBuf, 1);
(191)     printf("Sending Text Message: %s\n", pBuffer);
(192)
(193)     /* Publish the message. */
(194)     TopicPublisherPublish(pTopicPublisher, pTextMessage, &iErr,
(195)     szErrBuf);
(196)     check_error(iErr, szErrBuf, 1);
(197)
(198)     /* Commit the session. */
(199)     SessionCommit(pTopicSession, &iErr, szErrBuf);
(200)     check_error(iErr, szErrBuf, 1);
(201)
(202)     /* Close and clean up. */
(203)     TopicPublisherClose(pTopicPublisher, &iErr, szErrBuf);
(204)     check_error(iErr, szErrBuf, 1);
(205)     SessionClose(pTopicSession, &iErr, szErrBuf);
(206)     check_error(iErr, szErrBuf, 1);
(207)     ConnectionClose(pTopicConnection, &iErr, szErrBuf);
(208)     check_error(iErr, szErrBuf, 1);
(209)     DeleteMessage(pTextMessage, &iErr, szErrBuf);
(210)     DeleteTopicPublisher(pTopicPublisher, &iErr, szErrBuf);
(211)     DeleteSession(pTopicSession, &iErr, szErrBuf);
(212)     DeleteDestination(pTopic, &iErr, szErrBuf);
(213)     DeleteConnection(pTopicConnection, &iErr, szErrBuf);
(214)     DeleteTopicConnectionFactory(pTcf, &iErr, szErrBuf);
(215) }
(216)
(217)
(218) /*
(219) * =====
(220) * Topic Subscriber
(221) * This routine demonstrates how to subscribe a message from
(222) * a topic.
(223) * =====
(224) */
(225) static void subscriber() {
(226)     SBYN_Session*      pTopicSession;
(227)     SBYN_Destination*  pTopic;
(228)     SBYN_Message*      pReceivedMessage = 0;
(229)     SBYN_TopicSubscriber* pTopicSubscriber;
(230)     SBYN_Connection*   pTopicConnection;
```

```
(231)     SBYN_TopicConnectionFactory* pTcf;
(232)     unsigned long             ulMessageSize = 1024;
(233)     unsigned long             ulMessageCount = 10;
(234)     int                       iErr;
(235)     char                      szErrBuf[256];
(236)     char                      szUserInput[80];
(237)     char                      pTopicName[] = "eGatePubSubSample";
(238)
(239)     /* create a topic connection */
(240)     pTcf = CreateTopicConnectionFactory(spHostName, susPort, &iErr,
(241)     szErrBuf);
(242)     check_error(iErr, szErrBuf, 1);
(243)     if(!pTcf) {
(244)         printf("CreateTopicConnectionFactory
(245)     failed\nError:%0X\nReason:%s\n", iErr, szErrBuf);
(246)         exit(2);
(247)     }
(248)
(249)     /* create a topic connection */
(250)     pTopicConnection = CreateTopicConnection(pTcf,&iErr, szErrBuf);
(251)     check_error(iErr, szErrBuf, 1);
(252)
(253)     /* set client ID */
(254)     ConnectionSetClientID(pTopicConnection,
(255)     (char*)"TopicTestSubConnection",&iErr, szErrBuf);
(256)     check_error(iErr, szErrBuf, 1);
(257)
(258)     /* start connection */
(259)     ConnectionStart(pTopicConnection,&iErr, szErrBuf);
(260)     check_error(iErr, szErrBuf, 1);
(261)
(262)     /* create a topic session */
(263)     pTopicSession = ConnectionCreateTopicSession (pTopicConnection,
(264)     SBYN_TRANSACTED, SBYN_CLIENT_ACKNOWLEDGE, &iErr, szErrBuf);
(265)     check_error(iErr, szErrBuf, 1);
(266)     if(!pTopicSession) {
(267)         printf("CreateTopicSession failed\nError:%0X\nReason:%s\n",
(268)     iErr, szErrBuf);
(269)         exit(2);
(270)     }
(271)
(272)     /* create a topic */
(273)     pTopic = SessionCreateTopic(pTopicSession, pTopicName, &iErr,
(274)     szErrBuf);
(275)     check_error(iErr, szErrBuf, 1);
(276)
(277)     /* create a subscriber */
(278)     pTopicSubscriber = SessionCreateDurableSubscriber(pTopicSession,
(279)     pTopic, (char*)"TopicTestSubscriber",&iErr, szErrBuf);
(280)     check_error(iErr, szErrBuf, 1);
(281)
(282)     printf("Waiting for message ... \n");
(283)     do {
(284)         /* waiting for incoming messages */
(285)         pReceivedMessage = TopicSubscriberReceive(pTopicSubscriber,
(286)     &iErr, szErrBuf);
(287)         check_error(iErr, szErrBuf, 1);
(288)         if (pReceivedMessage->type == SBYN_MESSAGE_TYPE_TEXT) {
(289)             char *pReturnedBuf;
(290)             SBYN_WString *pWstr;
(291)             char* pMessageType;
(292)             /* retrieve the JMS message type */
(293)             pWstr = GetJMSType(pReceivedMessage, &iErr, szErrBuf);
(294)             check_error(iErr, szErrBuf, 1);
```

```

(295)         pMessageType = WStringToChar(pWstr);
(296)         check_error(iErr, szErrBuf, 1);
(297)
(298)         /* retrieve the text from message */
(299)         pReturnedBuf = GetText(pReceivedMessage, &iErr, szErrBuf);
(300)         printf("Received text message (JMSType:%s):  %s\n",
(301) pMessageType, pReturnedBuf);
(302)         free(pReturnedBuf);
(303)         DeleteWString(pWstr);
(304)         free((void*)pMessageType);
(305)         SessionCommit(pTopicSession, &iErr, szErrBuf);
(306)     }
(307)     printf("Enter 'r' for receiving more message, 'q' for
(308) exit\n");
(309)     scanf("%s", szUserInput);
(310) } while ( szUserInput[0] != 'q' && szUserInput[0] != 'Q' );
(311)
(312) check_error(iErr, szErrBuf, 1);
(313)
(314) /* close subscriber, session ... */
(315) TopicSubscriberClose(pTopicSubscriber, &iErr, szErrBuf);
(316) SessionClose(pTopicSession, &iErr, szErrBuf);
(317) ConnectionClose(pTopicConnection, &iErr, szErrBuf);
(318)
(319) /* delete objects */
(320) DeleteMessage(pReceivedMessage, &iErr, szErrBuf);
(321) DeleteDestination(pTopic, &iErr, szErrBuf);
(322) DeleteTopicSubscriber(pTopicSubscriber, &iErr, szErrBuf);
(323) DeleteSession(pTopicSession, &iErr, szErrBuf);
(324) DeleteConnection(pTopicConnection, &iErr, szErrBuf);
(325) DeleteTopicConnectionFactory(pTcf, &iErr, szErrBuf);
(326) }

```

4.5.2. Queue Messaging (Sending/Receiving) Using C

```

(1)  /* -----
(2)  *   Sample code to demonstrate JMS Queue Messaging using C.
(3)  *
(4)  * -----
(5)  *
(6)  * Disclaimer:
(7)  *
(8)  * Copyright 2002 by SeeBeyond Technology Corporation.
(9)  * All Rights Reserved.
(10) *
(11) * Unless otherwise stipulated in a written agreement for this software,
(12) * duly executed by SeeBeyond Technology Corporation, this software is
(13) * provided as is without warranty of any kind. The entire risk as to
(14) * the results and performance of this software is assumed by the user.
(15) * SeeBeyond Technology Corporation disclaims all warranties, either
(16) * express or implied, including but not limited, the implied warranties
(17) * of merchantability, fitness for a particular purpose, title and
(18) * non-infringement, with respect to this software.
(19) *
(20) * -----*/
(21)
(22) #include <mscapi.h>
(23) #include <stdlib.h>
(24) #include <stdio.h>
(25) #include <string.h>
(26) #ifndef WIN32
(27) #include <unistd.h>
(28) #endif
(29)

```

```
(30) #if defined(OS400)
(31) extern char *optarg;
(32) #endif
(33)
(34) #if defined(__gnu__)
(35) #include <getopt.h>
(36) #endif
(37)
(38) char          optionProducer[] = "[ -u ] [ -p port ]
(39)              [ -h hostname ]";
(40) char          optionConsumer[] = "[ -c ] [ -p port ]
(41)              [ -h hostname ]";
(42) char          optdescription[] = "\t-u run as a
(43)              producer\n\t-c run as a consumer\n\t-p port
(44)              number\n\t-h hostname\n";
(45) char*         spHostName;
(46) char          localhost[] = "localhost";
(47) static unsigned short susPort = 24053;
(48) int           iErr;
(49) char          szErrBuf[256];
(50)
(51) /* Routine for checking errors.*/
(52) static void check_error(int err, char* errBuf, int exitnow)
(53) {
(54)     if (err){
(55)         printf("ERROR:0x%x - %s\n", err, errBuf);
(56)         if (exitnow)
(57)             exit(1);
(58)     }
(59) }
(60)
(61) int main(int argc, char *argv[]) {
(62)     int          c;
(63)     char          cOption = 0;
(64)
(65)     spHostName = localhost;
(66)
(67)     while((c = getopt(argc, argv, ":p:h:P:H:cCUu")) != -1) {
(68)         switch(c){
(69)             case 'p':
(70)             case 'P':
(71)                 susPort = atoi(optarg); /* setup the port number */
(72)                 break;
(73)             case 'h':
(74)             case 'H':
(75)                 spHostName = optarg; /* setup the hostname */
(76)                 break;
(77)             case 'U':
(78)             case 'u':
(79)                 cOption = 'u'; /* run as a producer */
(80)                 break;
(81)             case 'c':
(82)             case 'C':
(83)                 cOption = 'c'; /* run as a consumer */
(84)                 break;
(85)             case ':':
(86)             case '?':
(87)                 printf("\nSYNOPSIS\n");
(88)                 printf("%s %s\n", argv[0], optionProducer);
(89)                 printf("%s %s\n", argv[0], optionConsumer);
(90)                 printf("%s\n", optdescription);
(91)                 exit(1);
(92)                 break;
(93)         }
```

```
(94)     }
(95)
(96)     if (cOption == 'u'){
(97)         sender();/* invoke producer */
(98)     } else if (cOption == 'c'){
(99)         receiver();/* invoke consumer */
(100)    } else {
(101)        printf("\nSYNOPSIS\n");
(102)        printf("%s %s\n", argv[0], optionProducer);
(103)        printf("%s %s\n", argv[0], optionConsumer);
(104)        printf("%s\n", optdescription);
(105)        exit(1);
(106)    }
(107) }
(108)
(109)
(110) void receiver() {
(111)     char                               pQueueName[] = "eGateP2PSample";
(112)     SBYN_QueueConnectionFactory*      pQcf = NULL;
(113)     SBYN_Connection*                  pQueueConnection = NULL;
(114)     SBYN_Session*                     pQueueSession = NULL;
(115)     SBYN_Destination*                 pQueue = NULL;
(116)     SBYN_QueueReceiver*               pQueueReceiver = NULL;
(117)     SBYN_Message*                     pMessage = NULL;
(118)     unsigned int                       iBufLen = 0;
(119)     char                               szUserInput[80];
(120)
(121)     printf("Queue name is %s\n", pQueueName);
(122)
(123)     /* Create a queue connection factory */
(124)     pQcf = CreateQueueConnectionFactory(spHostName, susPort, &iErr,
szErrBuf);
(125)     check_error(iErr, szErrBuf, 1);
(126)     if(!pQcf) {
(127)         printf("CreateQueueConnectionFactory failed\nError:%0X\nReason:%s\n",
iErr, szErrBuf);
(128)         exit(2);
(129)     }
(130)
(131)     /* Create a queue connection */
(132)     pQueueConnection = CreateQueueConnection(pQcf,&iErr, szErrBuf);
(133)     check_error(iErr, szErrBuf, 1);
(134)
(135)     /* Set the client ID */
(136)     ConnectionSetClientID(pQueueConnection, (char*)"RECEIVER", &iErr,
szErrBuf);
(137)     check_error(iErr, szErrBuf, 1);
(138)
(139)     /* Start the connection */
(140)     ConnectionStart(pQueueConnection,&iErr, szErrBuf);
(141)     check_error(iErr, szErrBuf, 1);
(142)
(143)     /* Create a queue session */
(144)     pQueueSession = ConnectionCreateQueueSession (pQueueConnection,
SBYN_TRANSACTED, SBYN_CLIENT_ACKNOWLEDGE, &iErr, szErrBuf);
(145)     check_error(iErr, szErrBuf, 1);
(146)     if(!pQueueSession) {
(147)         printf("CreateTopicSession failed\nError:%0X\nReason:%s\n", iErr,
szErrBuf);
(148)         exit(2);
(149)     }
(150)
(151)     /* Create a queue */
```

```
(152)     pQueue = SessionCreateQueue(pQueueSession, pQueueName, &iErr,
szErrBuf);
(153)     check_error(iErr, szErrBuf, 1);
(154)
(155)     /* Create a queue receiver */
(156)     pQueueReceiver = SessionCreateReceiver(pQueueSession, pQueue,
&iErr, szErrBuf);
(157)     check_error(iErr, szErrBuf, 1);
(158)
(159)     do {
(160)         /* Blocking for the message */
(161)         pMessage = QueueReceiverReceive(pQueueReceiver, &iErr, szErrBuf);
(162)         check_error(iErr, szErrBuf, 1);
(163)         if (pMessage->type == SBYN_MESSAGE_TYPE_TEXT) {
(164)             char *rtbuf;
(165)             rtbuf = GetText(pMessage, &iErr, szErrBuf);
(166)             printf("Received message: %s\n", rtbuf);
(167)             free(rtbuf);
(168)         }
(169)         printf("Enter 'r' for receiving more message, 'q' for exit\n");
(170)         scanf("%s", szUserInput);
(171)         } while ( szUserInput[0] != 'q' && szUserInput[0] != 'Q' );
(172)
(173)         /* now close the connections */
(174)         QueueReceiverClose(pQueueReceiver, &iErr, szErrBuf);
(175)         check_error(iErr, szErrBuf, 1);
(176)         SessionClose(pQueueSession, &iErr, szErrBuf);
(177)         check_error(iErr, szErrBuf, 1);
(178)         ConnectionStop(pQueueConnection, &iErr, szErrBuf);
(179)         check_error(iErr, szErrBuf, 1);
(180)         ConnectionClose(pQueueConnection, &iErr, szErrBuf);
(181)         check_error(iErr, szErrBuf, 1);
(182)
(183)         /* delete the objects */
(184)         DeleteMessage(pMessage, &iErr, szErrBuf);
(185)         DeleteQueueReceiver(pQueueReceiver, &iErr, szErrBuf);
(186)         DeleteDestination(pQueue, &iErr, szErrBuf);
(187)         DeleteSession(pQueueSession, &iErr, szErrBuf);
(188)         DeleteConnection(pQueueConnection, &iErr, szErrBuf);
(189)         DeleteQueueConnectionFactory(pQcf, &iErr, szErrBuf);
(190)     }
(191)
(192)
(193)
(194) void sender() {
(195)     char                pQueueName[] = "P2PSample";
(196)     SBYN_QueueConnectionFactory* pQcf = NULL;
(197)     SBYN_Connection*    pQueueConnection = NULL;
(198)     SBYN_Session*      pQueueSession = NULL;
(199)     SBYN_Destination*  pQueue = NULL;
(200)     SBYN_QueueSender*  pQueueSender = NULL;
(201)     SBYN_Message*      textMessage = NULL;
(202)     const int          MAX_MESSAGE_SIZE = 60;
(203)     char                pBuffer[] = "This is a text message";
(204)
(205)     /* Create a queue connection factory */
(206)     pQcf = CreateQueueConnectionFactory(spHostName, susPort, &iErr,
szErrBuf);
(207)     check_error(iErr, szErrBuf, 1);
(208)     if(!pQcf) {
(209)         printf("CreateQueueConnectionFactory failed\nError:%0X\nReason:%s\n",
iErr, szErrBuf);
(210)         exit(2);
(211)     }
```

```
(212)
(213)     /* Create a queue connection */
(214)     pQueueConnection = CreateQueueConnection(pQcf, &iErr, szErrBuf);
(215)     check_error(iErr, szErrBuf, 1);
(216)
(217)     /* Set the client ID */
(218)     ConnectionSetClientID(pQueueConnection, (char*)"SENDER", &iErr,
szErrBuf);
(219)     check_error(iErr, szErrBuf, 1);
(220)
(221)     /* Start the connection */
(222)     ConnectionStart(pQueueConnection, &iErr, szErrBuf);
(223)     check_error(iErr, szErrBuf, 1);
(224)
(225)     /* Create a queue session */
(226)     pQueueSession = ConnectionCreateQueueSession(pQueueConnection,
SBYN_TRANSACTED, SBYN_CLIENT_ACKNOWLEDGE, &iErr, szErrBuf);
(227)     check_error(iErr, szErrBuf, 1);
(228)     if(!pQueueSession) {
(229)         printf("CreateTopicSession failed\nError:%0X\nReason:%s\n", iErr,
szErrBuf);
(230)         exit(2);
(231)     }
(232)
(233)     /* Create a queue */
(234)     pQueue = SessionCreateQueue(pQueueSession, pQueueName, &iErr,
szErrBuf);
(235)     check_error(iErr, szErrBuf, 1);
(236)
(237)     /* Create a queue sender */
(238)     pQueueSender = SessionCreateSender(pQueueSession, pQueue, &iErr,
szErrBuf);
(239)     check_error(iErr, szErrBuf, 1);
(240)
(241)     /* Create a text message */
(242)     textMessage = SessionCreateTextMessage(pQueueSession, &iErr,
szErrBuf);
(243)     check_error(iErr, szErrBuf, 1);
(244)
(245)     /* set the mode to r/w */
(246)     ClearBody(textMessage, &iErr, szErrBuf);
(247)     check_error(iErr, szErrBuf, 1);
(248)
(249)     /* put in text */
(250)     SetText(textMessage, pBuffer, &iErr, szErrBuf);
(251)     check_error(iErr, szErrBuf, 1);
(252)     printf("Sending Message %s\n", pBuffer);
(253)
(254)     /* send out the message */
(255)     QueueSenderSend(pQueueSender, textMessage, &iErr, szErrBuf);
(256)     check_error(iErr, szErrBuf, 1);
(257)
(258)     /* session commit */
(259)     SessionCommit(pQueueSession, &iErr, szErrBuf);
(260)     check_error(iErr, szErrBuf, 1);
(261)
(262)     /* now close the connections */
(263)     QueueSenderClose(pQueueSender, &iErr, szErrBuf);
(264)     check_error(iErr, szErrBuf, 1);
(265)     SessionClose(pQueueSession, &iErr, szErrBuf);
(266)     check_error(iErr, szErrBuf, 1);
(267)     ConnectionStop(pQueueConnection, &iErr, szErrBuf);
(268)     check_error(iErr, szErrBuf, 1);
(269)     ConnectionClose(pQueueConnection, &iErr, szErrBuf);
```

```

(270)     check_error(iErr, szErrBuf, 1);
(271)
(272)     /* delete the objects */
(273)     DeleteMessage(textMessage, &iErr, szErrBuf);
(274)     DeleteQueueSender(pQueueSender, &iErr, szErrBuf);
(275)     DeleteDestination(pQueue, &iErr, szErrBuf);
(276)     DeleteSession(pQueueSession, &iErr, szErrBuf);
(277)     DeleteConnection(pQueueConnection, &iErr, szErrBuf);
(278)     DeleteQueueConnectionFactory(pQcf, &iErr, szErrBuf);
(279) }

```

4.5.3. Request-Reply Messaging Using C

```

(1)  /* -----
(2)  *   Sample code to demonstrate JMS Request-Reply messaging using C.
(3)  *
(4)  * -----
(5)  *
(6)  *   Disclaimer:
(7)  *
(8)  *   Copyright 2002 by SeeBeyond Technology Corporation.
(9)  *   All Rights Reserved.
(10) *
(11) *   Unless otherwise stipulated in a written agreement for this software,
(12) *   duly executed by SeeBeyond Technology Corporation, this software is
(13) *   provided as is without warranty of any kind. The entire risk as to
(14) *   the results and performance of this software is assumed by the user.
(15) *   SeeBeyond Technology Corporation disclaims all warranties, either
(16) *   express or implied, including but not limited, the implied warranties
(17) *   of merchantability, fitness for a particular purpose, title and
(18) *   non-infringement, with respect to this software.
(19) *
(20) * -----*/
(21)
(22) #include "mscapi.h"
(23) #include <stdlib.h>
(24) #include <stdio.h>
(25) #include <string.h>
(26)
(27) #ifndef WIN32
(28) #include <unistd.h>
(29) #endif
(30) #if defined(WIN32)
(31) #include <sbyn_getopt.h>
(32) #endif
(33)
(34) #if defined(OS400)
(35) extern char *optarg;
(36) #endif
(37)
(38) #if defined(__gnu__)
(39) #include <getopt.h>
(40) #endif
(41)
(42) static void requestor();
(43)
(44) /* Routine for checking errors.*/
(45) static void check_error(int err, char* errBuf, int exitnow)
(46) {
(47)     if (err){
(48)         printf("ERROR:0x%x - %s\n", err, errBuf);
(49)         if (exitnow)
(50)             exit(1);
(51)     }

```



```
(52) }
(53)
(54) char          optionRequestor[] = "[ -r ] [ -p port ]
(55)              [ -h hostname ]";
(56) char          optdescription[] = "\t-r run as a
(57)              requestor\n\t-p port number\n\t-h
(58)              hostname\n";
(59) char*         spHostName;
(60) char          localhost[] = "localhost";
(61) static unsigned short susPort = 24053;
(62) int           iErr;
(63) char          szErrBuf[256];
(64) char          pQueueName[] = "QueueRequestorSample";
(65)
(66) int main(int argc, char *argv[]) {
(67)     int         c;
(68)     char        cOption = 0;
(69)
(70)     spHostName = localhost;
(71)
(72)     while((c = getopt(argc, argv, ":p:h:P:H:rR")) != -1) {
(73)         switch(c){
(74)             case 'p':
(75)             case 'P':
(76)                 susPort = atoi(optarg); /* setup the port number */
(77)                 break;
(78)             case 'h':
(79)             case 'H':
(80)                 spHostName = optarg; /* setup the hostname */
(81)                 break;
(82)             case 'R':
(83)             case 'r':
(84)                 cOption = 'r'; /* run as a requestor */
(85)                 break;
(86)             case ':':
(87)             case '?':
(88)                 printf("\nSYNOPSIS\n");
(89)                 printf("%s %s\n", argv[0], optionRequestor);
(90)                 printf("%s\n", optdescription);
(91)                 exit(1);
(92)         }
(93)     }
(94)
(95)     if (cOption == 'r'){
(96)         requestor(); /* invoke requestor */
(97)     } else {
(98)         printf("\nSYNOPSIS\n");
(99)         printf("%s %s\n", argv[0], optionRequestor);
(100)        printf("%s\n", optdescription);
(101)        exit(1);
(102)    }
(103) }
(104)
(105)
(106)
(107) /*
(108) * =====
(109) * Queue Requestor
(110) * This routine demonstrates how to do request/reply.
(111) * =====
(112) */
(113) void requestor() {
(114)     SBYN_QueueConnectionFactory *pQcf = NULL;
(115)     SBYN_Connection *pQueueConnection = NULL;
```

```
(116)     SBYN_Session *pQueueSession           = NULL;
(117)     SBYN_Destination *pQueue            = NULL;
(118)     SBYN_QueueSender *pQueueSender      = NULL;
(119)     SBYN_Message *textMessage          = NULL;
(120)     SBYN_Message *pReplyMessage        = NULL;
(121)     SBYN_QueueRequestor *pQueueRequestor = 0;
(122)     char pBuffer[] = "This is a text message";
(123)
(124)     /* Create a queue connection factory */
(125)     pQcf = CreateQueueConnectionFactory(spHostName, susPort, &iErr,
szErrBuf);
(126)     check_error(iErr, szErrBuf, 1);
(127)     if(!pQcf) {
(128)         printf("CreateQueueConnectionFactory
failed\nError:%0X\nReason:%s\n", iErr, szErrBuf);
(129)         exit(2);
(130)     }
(131)
(132)     /* Create a queue connection */
(133)     pQueueConnection = CreateQueueConnection(pQcf,&iErr, szErrBuf);
(134)     check_error(iErr, szErrBuf, 1);
(135)
(136)     /* Set the client ID */
(137)     ConnectionSetClientID(pQueueConnection, (char*)"REQUESTOR",&iErr,
szErrBuf);
(138)     check_error(iErr, szErrBuf, 1);
(139)
(140)     /* Start the connection */
(141)     ConnectionStart(pQueueConnection,&iErr, szErrBuf);
(142)     check_error(iErr, szErrBuf, 1);
(143)
(144)     /* Create a queue session */
(145)     pQueueSession = ConnectionCreateQueueSession (pQueueConnection,
SBYN_NON_TRANSACTED, SBYN_CLIENT_ACKNOWLEDGE, &iErr, szErrBuf);
(146)     check_error(iErr, szErrBuf, 1);
(147)     if(!pQueueSession) {
(148)         printf("CreateTopicSession failed\nError:%0X\nReason:%s\n",
iErr, szErrBuf);
(149)         exit(2);
(150)     }
(151)
(152)     /* Create a queue */
(153)     pQueue = SessionCreateQueue(pQueueSession, pQueueName, &iErr,
szErrBuf);
(154)     check_error(iErr, szErrBuf, 1);
(155)
(156)     /* Create a queue requestor */
(157)     pQueueRequestor = CreateQueueRequestor(pQueueSession,pQueue,
&iErr, szErrBuf);
(158)     check_error(iErr, szErrBuf, 1);
(159)
(160)     /* Create a text message and make a request */
(161)     textMessage = SessionCreateTextMessage(pQueueSession, &iErr,
szErrBuf);
(162)     check_error(iErr, szErrBuf, 1);
(163)
(164)     /* set the mode to r/w */
(165)     ClearBody(textMessage, &iErr, szErrBuf);
(166)     check_error(iErr, szErrBuf, 1);
(167)
(168)     /* Copy in the text to be sent. */
(169)     SetText(textMessage, pBuffer, &iErr, szErrBuf);
(170)     check_error(iErr, szErrBuf, 1);
(171)     printf("Sending Message: %s\n", pBuffer);
```

```
(172)
(173)     /* Set ReplyTo destination */
(174)     SetJMSReplyTo(textMessage, pQueue, &iErr, szErrBuf);
(175)     check_error(iErr, szErrBuf, 1);
(176)
(177)     /* Make a request and wait for a reply */
(178)     pReplyMessage = QueueRequestorRequestTimeout(pQueueRequestor,
textMessage, 100000, &iErr, szErrBuf);
(179)     check_error(iErr, szErrBuf, 1);
(180)
(181)     /* Extract the message type */
(182)     if (GetMessageType(pReplyMessage, &iErr, szErrBuf) ==
SBYN_MESSAGE_TYPE_TEXT) {
(183)         char *rtbuf;
(184)         check_error(iErr, szErrBuf, 1);
(185)         /* Extract the text */
(186)         rtbuf = GetText(pReplyMessage, &iErr, szErrBuf);
(187)         check_error(iErr, szErrBuf, 1);
(188)         printf("Received message:  %s\n", rtbuf);
(189)         free(rtbuf);
(190)     }
(191)     DeleteMessage(pReplyMessage, &iErr, szErrBuf);
(192)     check_error(iErr, szErrBuf, 1);
(193)
(194)     /* now close the connections */
(195)     SessionClose(pQueueSession, &iErr, szErrBuf);
(196)     check_error(iErr, szErrBuf, 1);
(197)     ConnectionStop(pQueueConnection, &iErr, szErrBuf);
(198)     check_error(iErr, szErrBuf, 1);
(199)     ConnectionClose(pQueueConnection, &iErr, szErrBuf);
(200)     check_error(iErr, szErrBuf, 1);
(201)
(202)     /* delete the objects */
(203)     DeleteMessage(textMessage, &iErr, szErrBuf);
(204)     DeleteQueueSender(pQueueSender, &iErr, szErrBuf);
(205)     DeleteDestination(pQueue, &iErr, szErrBuf);
(206)     DeleteSession(pQueueSession, &iErr, szErrBuf);
(207)     DeleteConnection(pQueueConnection, &iErr, szErrBuf);
(208)     DeleteQueueConnectionFactory(pQcf, &iErr, szErrBuf);
(209) }
(210)
(211)
(212)
(213) void receiver() {
(214)     SBYN_QueueConnectionFactory* pQcf = NULL;
(215)     SBYN_Connection* pQueueConnection = NULL;
(216)     SBYN_Session* pQueueSession = NULL;
(217)     SBYN_Destination* pQueue = NULL;
(218)     SBYN_Destination* pReplyToQueue = NULL;
(219)     SBYN_QueueReceiver* pQueueReceiver = NULL;
(220)     SBYN_QueueSender* pReplyQueueSender = NULL;
(221)     SBYN_Message *pMessage = NULL;
(222)     char szUserInput[80];
(223)
(224)     printf("Queue name is %s\n", pQueueName);
(225)     pQcf = CreateQueueConnectionFactory(spHostName, susPort, &iErr,
szErrBuf);
(226)     if(!pQcf) {
(227)         printf("CreateQueueConnectionFactory
failed\nError:%0X\nReason:%s\n", iErr, szErrBuf);
(228)         exit(2);
(229)     }
(230)
(231)     pQueueConnection = CreateQueueConnection(pQcf, &iErr, szErrBuf);
```

```

(232)     ConnectionSetClientID(pQueueConnection, (char*)"RECEIVER",&iErr,
szErrBuf);
(233)     ConnectionStart(pQueueConnection,&iErr, szErrBuf);
(234)     pQueueSession = ConnectionCreateQueueSession(pQueueConnection,
SBYN_TRANSACTED, SBYN_CLIENT_ACKNOWLEDGE, &iErr, szErrBuf);
(235)     if(!pQueueSession) {
(236)         printf("CreateTopicSession failed\nError:%0X\nReason:%s\n",
iErr, szErrBuf);
(237)         exit(2);
(238)     }
(239)     pQueue = SessionCreateQueue(pQueueSession, pQueueName, &iErr,
szErrBuf);
(240)     pQueueReceiver = SessionCreateReceiver(pQueueSession, pQueue,
&iErr, szErrBuf);
(241)     do {
(242)
(243)         pMessage = QueueReceiverReceive(pQueueReceiver, &iErr,
szErrBuf);
(244)         if (pMessage->type == SBYN_MESSAGE_TYPE_TEXT) {
(245)             char *rtbuf;
(246)             rtbuf = GetText(pMessage, &iErr, szErrBuf);
(247)             printf("Reading message:  %s\n", rtbuf);
(248)             free(rtbuf);
(249)         } else {
(250)             printf("Error: Received invalid message format\n");
(251)         }
(252)         pReplyToQueue = GetJMSReplyTo(pMessage, &iErr, szErrBuf);
(253)         pReplyQueueSender = SessionCreateSender(pQueueSession,
pReplyToQueue, &iErr, szErrBuf);
(254)         ClearBody(pMessage, &iErr, szErrBuf);
(255)         SetText(pMessage, (char*)"This is reply message", &iErr,
szErrBuf);
(256)         QueueSenderSend(pReplyQueueSender, pMessage, &iErr, szErrBuf);
(257)         SessionCommit(pQueueSession, &iErr, szErrBuf);
(258)         printf("Enter 'r' for receiving more message, 'q' for
exit\n");
(259)         scanf("%s", szUserInput);
(260)     } while ( szUserInput[0] != 'q' && szUserInput[0] != 'Q' );
(261)
(262)     /* now close the connections */
(263)     QueueReceiverClose(pQueueReceiver, &iErr, szErrBuf);
(264)     SessionClose(pQueueSession, &iErr, szErrBuf);
(265)     ConnectionStop(pQueueConnection, &iErr, szErrBuf);
(266)     ConnectionClose(pQueueConnection, &iErr, szErrBuf);
(267)
(268)     /* delete the objects */
(269)     DeleteMessage(pMessage, &iErr, szErrBuf);
(270)     DeleteQueueReceiver(pQueueReceiver, &iErr, szErrBuf);
(271)     DeleteDestination(pQueue, &iErr, szErrBuf);
(272)     DeleteSession(pQueueSession, &iErr, szErrBuf);
(273)     DeleteConnection(pQueueConnection, &iErr, szErrBuf);
(274)     DeleteQueueConnectionFactory(pQcf, &iErr, szErrBuf);
(275) }

```

4.5.4. Message Selector Using C

```

(1)  /* -----
(2)  *   Sample code to demonstrate JMS Message Selectors using C.
(3)  *
(4)  * -----
(5)  *
(6)  *   Disclaimer:
(7)  *

```

```
(8)  *   Copyright 2002 by SeeBeyond Technology Corporation.
(9)  *   All Rights Reserved.
(10) *
(11) *   Unless otherwise stipulated in a written agreement for this software,
(12) *   duly executed by SeeBeyond Technology Corporation, this software is
(13) *   provided as is without warranty of any kind. The entire risk as to
(14) *   the results and performance of this software is assumed by the user.
(15) *   SeeBeyond Technology Corporation disclaims all warranties, either
(16) *   express or implied, including but not limited, the implied warranties
(17) *   of merchantability, fitness for a particular purpose, title and
(18) *   non-infringement, with respect to this software.
(19) *
(20) * -----*/
(21)
(22) #include <mscapi.h>
(23) #include <stdlib.h>
(24) #include <stdio.h>
(25) #include <string.h>
(26)
(27) #ifndef WIN32
(28) #include <unistd.h>
(29) #endif
(30) #if defined(WIN32)
(31) #include <sbyn_getopt.h>
(32) #endif
(33)
(34) #if defined(OS400)
(35) extern char *optarg;
(36) #endif
(37)
(38) #if defined(__gnu__)
(39) #include <getopt.h>
(40) #endif
(41)
(42) char          optionProducer[] = "[ -u ] [ -p port ]
(43)             [ -h hostname ]";
(44) char          optionConsumer[] = "[ -c ] [ -p port ]
(45)             [ -h hostname ]";
(46) char          optdescription[] = "\t-u run as a
(47)             producer\n\t-c run as a consumer\n\t-p
(48)             port number\n\t-h hostname\n";
(49) static char    localhost[] = "localhost";
(50) static unsigned short susPort = 24053; /* default port number */
(51) unsigned long  sulMessageSize = 16; /* default host name */
(52) static char*   spHostName;
(53) static char    PROP_NAME[] = "property";
(54) int            iErr;
(55) char          szErrBuf[256];
(56)
(57) static void selector_publisher();
(58) static void selector_subscriber();
(59)
(60) /* Check for errors. */
(61) static void check_error(int err, char* errBuf, int exitnow)
(62) {
(63)     if (err){
(64)         printf("ERROR:0x%x - %s\n", err, errBuf);
(65)         if (exitnow)
(66)             exit(1);
(67)     }
(68) }
(69)
(70)
(71) int main(int argc, char *argv[]) {
(72)     int            c;
```

```
(73)     char          cOption = 0;
(74)
(75)     spHostName = localhost;
(76)
(77)     while((c = getopt(argc, argv, ":p:h:P:H:cCUu")) != -1) {
(78)         switch(c) {
(79)             case 'p':
(80)             case 'P':
(81)                 susPort = atoi(optarg); /* setup the port number */
(82)                 break;
(83)             case 'h':
(84)             case 'H':
(85)                 spHostName = optarg; /* setup the hostname */
(86)                 break;
(87)             case 'U':
(88)             case 'u':
(89)                 cOption = 'u'; /* run as a producer */
(90)                 break;
(91)             case 'c':
(92)             case 'C':
(93)                 cOption = 'c'; /* run as a consumer */
(94)                 break;
(95)             case ':':
(96)             case '?':
(97)                 printf("\nSYNOPSIS\n");
(98)                 printf("%s %s\n", argv[0], optionProducer);
(99)                 printf("%s %s\n", argv[0], optionConsumer);
(100)                printf("%s\n", optdescription);
(101)                exit(1);
(102)                break;
(103)            }
(104)        }
(105)
(106)        if (cOption == 'u') {
(107)            selector_publisher(); /* invoke producer */
(108)        } else if (cOption == 'c') {
(109)            selector_subscriber(); /* invoke consumer */
(110)        } else {
(111)            printf("\nSYNOPSIS\n");
(112)            printf("%s %s\n", argv[0], optionProducer);
(113)            printf("%s %s\n", argv[0], optionConsumer);
(114)            printf("%s\n", optdescription);
(115)            exit(1);
(116)        }
(117)    }
(118)
(119) static void selector_publisher() {
(120)     SBYN_TopicConnectionFactory* pTcf;
(121)     SBYN_Connection* pTopicConnection = NULL;
(122)     SBYN_Session* pTopicSession = NULL;
(123)     SBYN_Destination* pTopic = NULL;
(124)     SBYN_TopicPublisher* pTopicPublisher = NULL;
(125)     int ii;
(126)     SBYN_Message msglist[10];
(127)     static char TOPIC_NAME[] = "Selector";
(128)
(129)     /* Create a topic factory. */
(130)     pTcf = CreateTopicConnectionFactory(spHostName, susPort, &iErr,
szErrBuf);
(131)     if(!pTcf) {
(132)         printf("CreateTopicConnectionFactory
failed\nError:%0X\nReason:%s\n", iErr, szErrBuf);
(133)         exit(2);
(134)     }
```

```
(135)
(136)     /* Create a topic connection. */
(137)     pTopicConnection = CreateTopicConnection(pTcf, &iErr, szErrBuf);
(138)     check_error(iErr, szErrBuf, 1);
(139)
(140)     /* Set the client ID. */
(141)     ConnectionSetClientID(pTopicConnection, (char*)"Publisher", &iErr,
szErrBuf);
(142)     check_error(iErr, szErrBuf, 1);
(143)
(144)     /* Start the connection. */
(145)     ConnectionStart(pTopicConnection, &iErr, szErrBuf);
(146)     check_error(iErr, szErrBuf, 1);
(147)
(148)     /* Create a topic session. */
(149)     pTopicSession = ConnectionCreateTopicSession (pTopicConnection,
SBYN_TRANSACTIONED, SBYN_CLIENT_ACKNOWLEDGE, &iErr, szErrBuf);
(150)     check_error(iErr, szErrBuf, 1);
(151)     if(!pTopicSession) {
(152)         printf("CreateTopicSession failed\nError:%0X\nReason:%s\n",
iErr, szErrBuf);
(153)         exit(2);
(154)     }
(155)
(156)     /* Create a topic. */
(157)     pTopic = SessionCreateTopic(pTopicSession, TOPIC_NAME, &iErr,
szErrBuf);
(158)     check_error(iErr, szErrBuf, 1);
(159)
(160)     /* Create a topic publisher. */
(161)     pTopicPublisher = SessionCreatePublisher(pTopicSession, pTopic,
&iErr, szErrBuf);
(162)     check_error(iErr, szErrBuf, 1);
(163)
(164)     /* Set delivery mode as persistent */
(165)     TopicPublisherSetDeliveryMode(pTopicPublisher, SBYN_PERSISTENT,
&iErr, szErrBuf);
(166)     check_error(iErr, szErrBuf, 1);
(167)
(168)     /* publish 10 messages to the topic */
(169)     for(ii=0; ii<10 ;ii++){
(170)         int index;
(171)         char buf[80];
(172)         /* Create a text message. */
(173)         msglist[ii].message = SessionCreateTextMessage(pTopicSession,
&iErr, szErrBuf);
(174)         /* Clear the body (payload) of the message. */
(175)         ClearBody((SBYN_Message*)msglist[ii].message, &iErr,
szErrBuf);
(176)         check_error(iErr, szErrBuf, 1);
(177)         msglist[ii].type = SBYN_MESSAGE_TYPE_TEXT;
(178)         index = ii%10;
(179)         sprintf(buf, "%d", index);
(180)         /* Set the string property */
(181)         SetStringProperty((SBYN_Message*)msglist[ii].message,
PROP_NAME, buf, &iErr, szErrBuf);
(182)         check_error(iErr, szErrBuf, 1);
(183)         /* Copy in the text to be sent. */
(184)         SetText((SBYN_Message*)msglist[ii].message, (char*)"This is a
text message", &iErr, szErrBuf);
(185)         check_error(iErr, szErrBuf, 1);
(186)         /* Publish the message. */
(187)         TopicPublisherPublish(pTopicPublisher,
(SBYN_Message*)msglist[ii].message, &iErr, szErrBuf);
```

```
(188)         check_error(iErr, szErrBuf, 1);
(189)         printf("... Published 1 message with property %s = %d\n",
PROP_NAME, ii);
(190)     }
(191)     /* Commit the session. */
(192)     SessionCommit(pTopicSession, &iErr, szErrBuf);
(193)     check_error(iErr, szErrBuf, 1);
(194)
(195)     /* close and delete objects */
(196)     TopicPublisherClose(pTopicPublisher, &iErr, szErrBuf);
(197)     check_error(iErr, szErrBuf, 1);
(198)     SessionClose(pTopicSession, &iErr, szErrBuf);
(199)     check_error(iErr, szErrBuf, 1);
(200)     ConnectionClose(pTopicConnection, &iErr, szErrBuf);
(201)     check_error(iErr, szErrBuf, 1);
(202)     DeleteTopicPublisher(pTopicPublisher, &iErr, szErrBuf);
(203)     DeleteSession(pTopicSession, &iErr, szErrBuf);
(204)     DeleteConnection(pTopicConnection, &iErr, szErrBuf);
(205)     DeleteTopicConnectionFactory(pTcf, &iErr, szErrBuf);
(206) }
(207)
(208)
(209) static void selector_subscriber() {
(210)     SBYN_TopicConnectionFactory *pTcf;
(211)     SBYN_Connection *pTopicConnection = 0;
(212)     SBYN_Session *pTopicSession = 0;
(213)     SBYN_Destination *topic = 0;
(214)     SBYN_TopicSubscriber *pTopicSubscriber = 0;
(215)     SBYN_Message *pMessage = 0;
(216)     char selectorString[80];
(217)     char selectorSubscriberName[80];
(218)     int selector = 7;
(219)     char* selectorName;
(220)     static char TOPIC_NAME[] = "eGateSelector";
(221)
(222)
(223)     /* create a topic connection */
(224)     pTcf = CreateTopicConnectionFactory(spHostName, susPort, &iErr,
szErrBuf);
(225)     check_error(iErr, szErrBuf, 1);
(226)     if(!pTcf) {
(227)         printf("CreateTopicConnectionFactory
failed\nError:%0X\nReason:%s\n", iErr, szErrBuf);
(228)         exit(2);
(229)     }
(230)
(231)     /* create a topic connection */
(232)     pTopicConnection = CreateTopicConnection(pTcf, &iErr, szErrBuf);
(233)     check_error(iErr, szErrBuf, 1);
(234)
(235)     /* set client ID */
(236)     ConnectionSetClientID(pTopicConnection, (char*)"Publisher", &iErr,
szErrBuf);
(237)     check_error(iErr, szErrBuf, 1);
(238)
(239)     /* start connection */
(240)     ConnectionStart(pTopicConnection, &iErr, szErrBuf);
(241)     check_error(iErr, szErrBuf, 1);
(242)
(243)     /* create a topic session */
(244)     pTopicSession = ConnectionCreateTopicSession(pTopicConnection,
SBYN_TRANSACTED, SBYN_CLIENT_ACKNOWLEDGE, &iErr, szErrBuf);
(245)     check_error(iErr, szErrBuf, 1);
(246)     if(!pTopicSession) {
```



```
(247)         printf("CreateTopicSession failed\nError:%0X\nReason:%s\n",
iErr, szErrBuf);
(248)         exit(2);
(249)     }
(250)
(251)     /* create a topic */
(252)     topic = SessionCreateTopic(pTopicSession, TOPIC_NAME, &iErr,
szErrBuf);
(253)     check_error(iErr, szErrBuf, 1);
(254)
(255)     /* create subscriber with selector*/
(256)     sprintf(selectorString, "%s = '%d'", PROP_NAME, selector);
(257)     selectorString[strlen(selectorString)] = '\0';
(258)     sprintf(selectorSubscriberName, "SelectorSubscriber%d", selector);
(259)     pTopicSubscriber =
SessionCreateDurableSubscriberMessageSelector(pTopicSession, topic,
selectorSubscriberName, selectorString, 0, &iErr, szErrBuf);
(260)     check_error(iErr, szErrBuf, 1);
(261)
(262)     /* Get message using selector */
(263)     selectorName = TopicSubscriberGetMessageSelector(pTopicSubscriber,
&iErr, szErrBuf);
(264)     check_error(iErr, szErrBuf, 1);
(265)     printf("using selector: %s\n", selectorName);
(266)     for (pMessage = TopicSubscriberReceive(pTopicSubscriber, &iErr,
szErrBuf);
(267)         pMessage != 0;
(268)         pMessage = TopicSubscriberReceiveTimeout(pTopicSubscriber,
1000, &iErr, szErrBuf))
(269)     {
(270)         char* property = WStringToChar(GetStringProperty(pMessage,
PROP_NAME, &iErr, szErrBuf));
(271)         printf("Received 1 message with %s = %s\n", PROP_NAME,
property);
(272)     }
(273)     check_error(iErr, szErrBuf, 1);
(274)
(275)     /* Session commit */
(276)     SessionCommit(pTopicSession, &iErr, szErrBuf);
(277)     check_error(iErr, szErrBuf, 1);
(278)
(279)     /* close and delete objects */
(280)     TopicSubscriberClose(pTopicSubscriber, &iErr, szErrBuf);
(281)     check_error(iErr, szErrBuf, 1);
(282)     SessionClose(pTopicSession, &iErr, szErrBuf);
(283)     check_error(iErr, szErrBuf, 1);
(284)     ConnectionClose(pTopicConnection, &iErr, szErrBuf);
(285)     check_error(iErr, szErrBuf, 1);
(286)
(287)     DeleteTopicSubscriber(pTopicSubscriber, &iErr, szErrBuf);
(288)     DeleteSession(pTopicSession, &iErr, szErrBuf);
(289)     DeleteConnection(pTopicConnection, &iErr, szErrBuf);
(290)     DeleteTopicConnectionFactory(pTcf, &iErr, szErrBuf);
(291) }
```

4.5.5. Publish/Subscribe Messaging Using XA in C

```
(1) *-----
(2) * Sample code to demonstrate JMS Pub/Sub using XA.
(3) *-----
(4) *
(5) * Disclaimer:
(6) *
(7) * Copyright 2002 by SeeBeyond Technology Corporation.
```

```
(8)  * All Rights Reserved.
(9)  * Unless otherwise stipulated in a written agreement for this
(10) * software, duly executed by SeeBeyond Technology Corporation, this
(11) * software is provided as is without warranty of any kind. The
(12) * entire risk as to the results and performance of this software
(13) * is assumed by the user. SeeBeyond Technology Corporation disclaims
(14) * all warranties, either express or implied, including but not
(15) * limited, the implied warranties of merchantability, fitness for a
(16) * particular purpose, title and non-infringement, with respect to
(17) * this software.
(18) *
(19) * -----*/
(20) #include "mscapi.h"
(21) #include <stdlib.h>
(22) #include <stdio.h>
(23) #include <string.h>
(24)
(25) #ifndef WIN32
(26) #include <unistd.h>
(27) #endif
(28) #if defined(WIN32)
(29) #include "sbyn_getopt.h"
(30) #endif
(31)
(32) #if defined(OS400)
(33) extern char *optarg;
(34) #endif
(35)
(36) #if defined(__gnu__)
(37) #include <getopt.h>
(38) #endif
(39)
(40)
(41) char          optionProducer[] = "[ -u ] [ -p port ]
(42)             [ -h hostname ]";
(43) char          optionConsumer[] = "[ -c ] [ -p port ]
(44)             [ -h hostname ]";
(45) char          optdescription[] = "\t-u run as a
(46) producer\n\t-c run as a consumer\n\t-p port
(47) number\n\t-h hostname\n";
(48) static char    localhost[] = "localhost";
(49) static unsigned short susPort = 24053; /* default port number */
(50) unsigned long  sulMessageSize = 16; /* default host name */
(51) static char*   spHostName;
(52) static int     iErr;
(53) static char    szErrBuf[256];
(54) static int     iNumMessages = 10;
(55) static char    szText[] = "This is a text message";
(56)
(57) static void XATopicPub();
(58) static void XATopicSub();
(59)
(60) #define MSCLIENT_DLL_NAME "stc_msclient.dll"
(61)
(62) /* Check for errors. */
(63) static void check_error(int err, char* errBuf, int exitnow)
(64) {
(65)     if (err){
(66)         printf("ERROR:0x%x - %s\n", err, errBuf);
(67)         if (exitnow)
(68)             exit(1);
(69)     }
(70) }
(71)
```

```
(72)
(73) int main(int argc, char *argv[]) {
(74)     int          c;
(75)     char          cOption = 0;
(76)
(77)     spHostName = localhost;
(78)
(79)     while((c = getopt(argc, argv, ":p:h:P:H:cCUu")) != -1) {
(80)         switch(c){
(81)             case 'p':
(82)             case 'P':
(83)                 susPort = atoi(optarg); /* setup the port number */
(84)                 break;
(85)             case 'h':
(86)             case 'H':
(87)                 spHostName = optarg; /* setup the hostname */
(88)                 break;
(89)             case 'U':
(90)             case 'u':
(91)                 cOption = 'u'; /* run as a producer */
(92)                 break;
(93)             case 'c':
(94)             case 'C':
(95)                 cOption = 'c'; /* run as a consumer */
(96)                 break;
(97)             case ':':
(98)             case '?':
(99)                 printf("\nSYNOPSIS\n");
(100)                printf("%s %s\n", argv[0], optionProducer);
(101)                printf("%s %s\n", argv[0], optionConsumer);
(102)                printf("%s\n", optdescription);
(103)                exit(1);
(104)                break;
(105)            }
(106)        }
(107)
(108)        if (cOption == 'u'){
(109)            XATopicPub(); /* invoke producer */
(110)        } else if (cOption == 'c'){
(111)            XATopicSub(); /* invoke consumer */
(112)        } else {
(113)            printf("\nSYNOPSIS\n");
(114)            printf("%s %s\n", argv[0], optionProducer);
(115)            printf("%s %s\n", argv[0], optionConsumer);
(116)            printf("%s\n", optdescription);
(117)            exit(1);
(118)        }
(119)    }
(120)
(121) /*
(122) * =====
(123) * Publish Message
(124) * This routine publishes iNumMessages to the topic
(125) * =====
(126) */
(127) static void PublishMessage(SBYN_TopicPublisher* pPublisher,
(128) SBYN_Message* pMessage, int iNumMessages)
(129) {
(130)     int ii;
(131)     for ( ii = 0; ii < iNumMessages; ii++){
(132)         SetIntProperty(pMessage, (char*)"Sequence", ii, &iErr,
(133) szErrBuf);
(134)         check_error(iErr, szErrBuf, 1);
(135)         printf("Sending Message: Sequence number %d\n", ii);
```

```
(136)         TopicPublisherPublish(pPublisher, pMessage, &iErr, szErrBuf);
(137)         check_error(iErr, szErrBuf, 1);
(138)     }
(139) }
(140)
(141)
(142) /*
(143) * =====
(144) * Receive Message
(145) * This routine block on receiving message for maximum iWait
(146) *   seconds before return.
(147) * =====
(148) */
(149) static int SubscriberReceiveMessage(SBYN_TopicSubscriber* pSub)
(150) {
(151)     int iMsgCount = 0;
(152)     SBYN_Message* pRMsg = 0;
(153)     char szUserInput[8];
(154)     printf("Waiting for message ... \n");
(155)     do {
(156)         pRMsg = TopicSubscriberReceive(pSub, &iErr, szErrBuf);
(157)         printf("Received Message %d\n", iMsgCount);
(158)         check_error(iErr, szErrBuf, 1);
(159)         iMsgCount++;
(160)         if (iMsgCount >= iNumMessages){
(161)             printf("Enter 'r' for receiving more message, 'q' for
(162)             exit\n");
(163)             scanf("%s", szUserInput);
(164)             iMsgCount = 0;
(165)         }
(166)
(167)     } while(szUserInput[0] != 'q' && szUserInput[0] != 'Q');
(168)     return iMsgCount;
(169) }
(170)
(171) /*
(172) * =====
(173) * Topic Publisher
(174) * This routine demonstrates how to publish to a topic.
(175) * =====
(176) */
(177) void XATopicPub()
(178) {
(179)     SBYN_XATopicConnectionFactory* pXATcf = 0;
(180)     SBYN_Connection* pConnection = 0;
(181)     SBYN_Session* pXATopicSession = 0;
(182)     SBYN_Session* pTopicSession = 0;
(183)     SBYN_Destination* pTopic = 0;
(184)     SBYN_XAResource* pXATopicResource;
(185)     SBYN_XAResource* pXATopicResourceTmp;
(186)     SBYN_TopicPublisher* pTopicPublisher;
(187)     SBYN_Message* pMessage;
(188)     SBYN_Xid* pXid;
(189)     char pTopicName[] = "XAPubSubSample";
(190)
(191)     /* create XA connection factory */
(192)     pXATcf = CreateXATopicConnectionFactory(spHostName, susPort, &iErr,
(193)     szErrBuf);
(194)     check_error(iErr, szErrBuf, 1);
(195)
(196)     /* create XA connection */
(197)     pConnection = CreateXATopicConnection(pXATcf, &iErr, szErrBuf);
(198)     check_error(iErr, szErrBuf, 1);
(199)
```

```
(200)      /* set client ID */
(201)      ConnectionSetClientID(pConnection, (char*)"eGate{7E527692-770A-
(202)      11D5-B139-935EB6E85DBD}", &iErr, szErrBuf);
(203)      check_error(iErr, szErrBuf, 1);
(204)
(205)      /* create XA session */
(206)      pXATopicSession = XAConnectionCreateXATopicSession(pConnection,
(207)      &iErr, szErrBuf);
(208)      check_error(iErr, szErrBuf, 1);
(209)
(210)      /* get session */
(211)      pTopicSession = XASessionGetTopicSession(pXATopicSession, &iErr,
(212)      szErrBuf);
(213)      check_error(iErr, szErrBuf, 1);
(214)
(215)      /* get XA resource */
(216)      pXATopicResource = XASessionGetXAResource(pXATopicSession, &iErr,
(217)      szErrBuf);
(218)      check_error(iErr, szErrBuf, 1);
(219)
(220)      /* get XA resource */
(221)      pXATopicResourceTmp = XASessionGetXAResource(pXATopicSession,
(222)      &iErr, szErrBuf);
(223)      check_error(iErr, szErrBuf, 1);
(224)
(225)      /* create a Topic */
(226)      pTopic = SessionCreateTopic(pTopicSession, pTopicName, &iErr,
(227)      szErrBuf);
(228)      check_error(iErr, szErrBuf, 1);
(229)
(230)      /* create a publisher */
(231)      pTopicPublisher = SessionCreatePublisher(pTopicSession, pTopic,
(232)      &iErr, szErrBuf);
(233)      check_error(iErr, szErrBuf, 1);
(234)
(235)      /* connection start */
(236)      ConnectionStart(pConnection, &iErr, szErrBuf);
(237)      check_error(iErr, szErrBuf, 1);
(238)
(239)      /* create xa id */
(240)      pXid = XACreateXid((char*)MSCLIENT_DLL_NAME, &iErr, szErrBuf);
(241)      check_error(iErr, szErrBuf, 1);
(242)
(243)      /* associate the global transaction with the resource */
(244)      XAResourceStart(pXATopicResource, pXid, SBYN_TMNOFLAGS, &iErr,
(245)      szErrBuf);
(246)      check_error(iErr, szErrBuf, 1);
(247)
(248)      /* create a meessage */
(249)      pMessage = SessionCreateTextMessage(pXATopicSession, &iErr,
(250)      szErrBuf);
(251)      check_error(iErr, szErrBuf, 1);
(252)
(253)      /* set mode to r/w */
(254)      ClearBody(pMessage, &iErr, szErrBuf);
(255)      check_error(iErr, szErrBuf, 1);
(256)
(257)      /* write bytes */
(258)      SetText(pMessage, (char*)szText, &iErr, szErrBuf);
(259)      check_error(iErr, szErrBuf, 1);
(260)
(261)      /* publish message */
(262)      printf("Sending %d messages\n", iNumMessages);
(263)      PublishMessage(pTopicPublisher, pMessage, iNumMessages);
```

```
(264)
(265)     /* xaEnd */
(266)     XAResourceEnd(pXATopicResource, pXid, SBYN_TMSUCCESS, &iErr,
(267)     szErrBuf);
(268)     check_error(iErr, szErrBuf, 1);
(269)
(270)     /* =====
(271)     * Prepare-Rollback
(272)     * =====
(273)     */
(274)     /* xaPrepare */
(275)     XAResourcePrepare(pXATopicResource, pXid, &iErr, szErrBuf);
(276)     check_error(iErr, szErrBuf, 1);
(277)
(278)     /* xaRollBack */
(279)     printf("Rolling back %d message\n", iNumMessages);
(280)     XAResourceRollback(pXATopicResource, pXid, &iErr, szErrBuf);
(281)     check_error(iErr, szErrBuf, 1);
(282)
(283)
(284)     /* =====
(285)     * Prepare-Commit
(286)     * =====
(287)     */
(288)
(289)     /* xa start */
(290)     XAResourceStart(pXATopicResource, pXid, SBYN_TMNOFLAGS, &iErr,
(291)     szErrBuf);
(292)     check_error(iErr, szErrBuf, 1);
(293)
(294)     /* send message */
(295)     printf("Sending %d messages\n", iNumMessages);
(296)     PublishMessage(pTopicPublisher, pMessage, iNumMessages);
(297)
(298)     /* xaEnd */
(299)     XAResourceEnd(pXATopicResource, pXid, SBYN_TMSUCCESS, &iErr,
(300)     szErrBuf);
(301)     check_error(iErr, szErrBuf, 1);
(302)
(303)     /* xaPrepare */
(304)     if (SBYN_XA_OK != XAResourcePrepare(pXATopicResource, pXid, &iErr,
(305)     szErrBuf))
(306)     {
(307)         printf("ERROR: XAResourcePrepare failed\n");
(308)     }
(309)     check_error(iErr, szErrBuf, 1);
(310)
(311)     /* xa commit */
(312)     printf("Session Commit...\n");
(313)     XAResourceCommit(pXATopicResource, pXid, TRUE, &iErr, szErrBuf);
(314)     check_error(iErr, szErrBuf, 1);
(315)
(316)     /* Close and clean up. */
(317)     TopicPublisherClose(pTopicPublisher, &iErr, szErrBuf);
(318)     check_error(iErr, szErrBuf, 1);
(319)     SessionClose(pXATopicSession, &iErr, szErrBuf);
(320)     check_error(iErr, szErrBuf, 1);
(321)     ConnectionClose(pConnection, &iErr, szErrBuf);
(322)     check_error(iErr, szErrBuf, 1);
(323)     DeleteMessage(pMessage, &iErr, szErrBuf);
(324)     DeleteTopicPublisher(pTopicPublisher, &iErr, szErrBuf);
(325)     DeleteXAResource(pXATopicResource, &iErr, szErrBuf);
(326)     DeleteXid(pXid, &iErr, szErrBuf);
(327)     DeleteSession(pXATopicSession, &iErr, szErrBuf); /* delete session
```

```
(328)     & resource */
(329)     DeleteDestination(pTopic, &iErr, szErrBuf);
(330)     DeleteConnection(pConnection, &iErr, szErrBuf);
(331)     DeleteXATopicConnectionFactory(pXATcf, &iErr, szErrBuf);
(332) }
(333)
(334) /*
(335) * =====
(336) * Topic Subscriber
(337) * This routine demonstrates how to subscribe a message from a
(338) * topic.
(339) * =====
(340) */
(341) void XATopicSub()
(342) {
(343)     SBYN_XATopicConnectionFactory* pXATcf = 0;
(344)     SBYN_Connection* pConnection = 0;
(345)     SBYN_Session* pXATopicSession = 0;
(346)     SBYN_Session* pTopicSession = 0;
(347)     SBYN_XAResource* pTopicResource = 0;
(348)     SBYN_Destination* pTopic = 0;
(349)     SBYN_Message* pReceivedMessage = 0;
(350)     SBYN_XAResource* pXATopicResource = 0;
(351)     SBYN_TopicSubscriber* pTopicSubscriber = 0;
(352)     SBYN_Message* pMessage = 0;
(353)     SBYN_Xid* pXid = 0;
(354)     char pTopicName[] = "eGateXAPubSubSample";
(355)     int iNumReceived = 0;
(356)     char szUserInput[8];
(357)     /* create XA connection factory */
(358)     pXATcf = CreateXATopicConnectionFactory(spHostName, susPort, &iErr,
(359)     szErrBuf);
(360)     check_error(iErr, szErrBuf, 1);
(361)
(362)     /* create XA connection */
(363)     pConnection = CreateXATopicConnection(pXATcf, &iErr, szErrBuf);
(364)     check_error(iErr, szErrBuf, 1);
(365)
(366)     /* set client ID */
(367)     ConnectionSetClientID(pConnection, (char*)"eGate{7E527692-770A-
(368)     11D5-B139-3456789}", &iErr, szErrBuf);
(369)     check_error(iErr, szErrBuf, 1);
(370)
(371)     /* create XA session */
(372)     pXATopicSession = XAConnectionCreateXATopicSession(pConnection,
(373)     &iErr, szErrBuf);
(374)     check_error(iErr, szErrBuf, 1);
(375)
(376)     /* get session */
(377)     pTopicSession = XASessionGetTopicSession(pXATopicSession, &iErr,
(378)     szErrBuf);
(379)     check_error(iErr, szErrBuf, 1);
(380)
(381)     /* get XA resource */
(382)     pXATopicResource = XASessionGetXAResource(pXATopicSession, &iErr,
(383)     szErrBuf);
(384)     check_error(iErr, szErrBuf, 1);
(385)
(386)     /* create a Topic */
(387)     pTopic = SessionCreateTopic(pTopicSession, pTopicName, &iErr,
(388)     szErrBuf);
(389)     check_error(iErr, szErrBuf, 1);
(390)
(391)     /* create a subscriber */
```

```
(392) //pTopicSubscriber = SessionCreateDurableSubscriber(pTopicSession,
(393) pTopic, (char*)"XATopicSubscriber", &iErr, szErrBuf);
(394) pTopicSubscriber = SessionCreateSubscriber(pTopicSession, pTopic,
(395) &iErr, szErrBuf);
(396) check_error(iErr, szErrBuf, 1);
(397)
(398) /* connection start */
(399) ConnectionStart(pConnection, &iErr, szErrBuf);
(400) check_error(iErr, szErrBuf, 1);
(401)
(402) /* start xa resource */
(403) pXid = XACreateXid((char*)MSCLIENT_DLL_NAME, &iErr, szErrBuf);
(404) check_error(iErr, szErrBuf, 1);
(405)
(406)
(407)
(408) /* Receive all the messages on the topic and return the number of
(409) messages received */
(410) //iNumReceived = SubscriberReceiveMessage(pTopicSubscriber);
(411) printf("Receiving messages...\n");
(412) do {
(413) int iMsgCount = 0;
(414)     SBYN_Message* pRMsg = 0;
(415)     /* associate the global transaction with the resource */
(416)     XAResourceStart(pXATopicResource, pXid, SBYN_TMNOFLAGS, &iErr,
(417)     szErrBuf);
(418)     check_error(iErr, szErrBuf, 1);
(419)     while(iMsgCount < iNumMessages){
(420)         pRMsg = TopicSubscriberReceive(pTopicSubscriber, &iErr,
(421)         szErrBuf);
(422)         printf("Received Message %d\n", iMsgCount);
(423)         iMsgCount++;
(424)     }
(425)
(426)     /* xaEnd */
(427)     XAResourceEnd(pXATopicResource, pXid, SBYN_TMSUCCESS, &iErr,
(428)     szErrBuf);
(429)     check_error(iErr, szErrBuf, 1);
(430)     XAResourceCommit(pXATopicResource, pXid, TRUE, &iErr,
(431)     szErrBuf);
(432)     printf("Enter 'r' for receiving more message, 'q' for
(433)     exit\n");
(434)     scanf("%s", szUserInput);
(435)     iMsgCount = 0;
(436) } while(szUserInput[0] != 'q' && szUserInput[0] != 'Q');
(437)
(438) /* Session commit*/
(439) SessionCommit(pTopicSession, &iErr, szErrBuf);
(440)
(441) /* close and delete objects */
(442) TopicSubscriberClose(pTopicSubscriber, &iErr, szErrBuf);
(443) SessionUnsubscribe(pXATopicSession, (char*)"TopicSubscriber",
(444) &iErr, szErrBuf);
(445) SessionClose(pXATopicSession, &iErr, szErrBuf);
(446) ConnectionClose(pConnection, &iErr, szErrBuf);
(447)
(448) /* delete objects */
(449) DeleteDestination(pTopic, &iErr, szErrBuf);
(450) DeleteTopicSubscriber(pTopicSubscriber, &iErr, szErrBuf);
(451) DeleteXAResource(pXATopicResource, &iErr, szErrBuf);
(452) DeleteXid(pXid, &iErr, szErrBuf);
(453) DeleteSession(pXATopicSession, &iErr, szErrBuf); /* delete XA
(454) resource and session */
(455) DeleteConnection(pConnection, &iErr, szErrBuf);
```



```
(456)     DeleteXATopicConnectionFactory(pXATcf, &iErr, szErrBuf);  
(457) }
```

4.6 Implementing JMS Models in C++

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

- **Session**—Characterized by a hostname, port number, session type (either a pub/sub “topic session” or a point-to-point “queue session”), and connection parameters such as acknowledgment mode, transaction mode, delivery mode, and client ID.
- **Destination**—Either a topic (for pub/sub messaging) or else a queue (for point-to-point messaging). Characterized by session and destination name.
- **Message Producer**—Either a topic publisher or a queue sender. Characterized by session and destination.
- **Message Consumer**—Either a topic subscriber or a queue receiver. Characterized by session and destination (and, in pub/sub messaging only, by the name of the durable subscriber, if designated).
- **Requestor**—In request/reply messaging, either a topic requestor or a queue requestor. Characterized by session, destination, message, and time-to-live value expressed in milliseconds.
- **Message**—Characterized by the message type and payload. The payload is also called the message *body*, or message *content*:
 - ♦ For messages of type `BytesMessage`, the payload is an array of bytes.
 - ♦ For messages of type `TextMessage`, the payload is a string of text characters. Native encoding is used; for example, text on AS/400 systems uses EBCDIC encoding, but ASCII is used on most other systems.

4.7 Sample Code for Using JMS in C++

From the **Samples** directory located on the e*Gate installation CD-ROM, navigate to the **jmsapi** folder. Select the **C** folder and extract the sample files from this folder to the computer that you have installed the e*Gate API Kit on. The samples were built using Microsoft Visual Studio 6.0.

For the JMS API in C++, the following sample programs are provided on the product CD-ROM:

- [Publish/Subscribe Messaging Using C++](#) on page 175
- [Queue Messaging \(Sending/Receiving\) Using C++](#) on page 179
- [Request-Reply Messaging Using C++](#) on page 183

- [Message Selector Using C++](#) on page 186
- [XA Publish/Subscribe Messaging For JMS Using C++](#) on page 190

4.7.1. Publish/Subscribe Messaging Using C++

```

(1)  *-----*
(2)  * Sample code to demonstrate JMS Pub/Sub.
(3)  *-----*
(4)  *
(5)  * Disclaimer:
(6)  *
(7)  * Copyright 2002 by SeeBeyond Technology Corporation.
(8)  * All Rights Reserved.
(9)  * Unless otherwise stipulated in a written agreement for this
(10) * software, duly executed by SeeBeyond Technology Corporation, this
(11) * software is provided as is without warranty of any kind. The
(12) * entire risk as to the results and performance of this software
(13) * is assumed by the user. SeeBeyond Technology Corporation disclaims
(14) * all warranties, either express or implied, including but not
(15) * limited, the implied warranties of merchantability, fitness for a
(16) * particular purpose, title and non-infringement, with respect to
(17) * this software.
(18) *
(19) *-----*/
(20) #include <ms.h>
(21) #include <mslocale.h>
(22)
(23) #define MSCLIENT_DLL_NAME "stc_msclient.dll"
(24) char option1[] = "[ -u ] [ -p port ] [ -h hostname ]";
(25) char option2[] = "[ -c ] [ -p port ] [ -h hostname ]";
(26) char optdescription[] = "\t-u run as a
(27) producer\n\t-c run as a consumer\n\t-p port
(28) number\n\t-h hostname\n";
(29) static char localhost[] = "localhost";
(30) static unsigned short susPort = 24053; /* default port number */
(31) unsigned long sulMessageSize = 16; /* default host name */
(32) static char* spHostName;
(33) static void subscriber();
(34) static void publisher();
(35)
(36)
(37) #ifndef WIN32
(38) #include <unistd.h>
(39) #endif
(40) #if defined(WIN32)
(41) #include "sbyn_getopt.h"
(42) #endif
(43)
(44) #if defined(OS400)
(45) extern char *optarg;
(46) #endif
(47)
(48) #if defined(__gnu__)
(49) #include <getopt.h>
(50) #endif
(51)
(52)
(53) int main(int argc, char *argv[]) {
(54)     int c;
(55)     char cOption = 0;
(56)
(57)     spHostName = localhost;

```

```
(58)
(59) while((c = getopt(argc, argv, ":p:h:P:H:cCUu")) != -1) {
(60)     switch(c) {
(61)         case 'p':
(62)         case 'P':
(63)             susPort = atoi(optarg); /* setup the port number */
(64)             break;
(65)         case 'h':
(66)         case 'H':
(67)             spHostName = optarg; /* setup the hostname */
(68)             break;
(69)         case 'U':
(70)         case 'u':
(71)             cOption = 'u'; /* run as a producer */
(72)             break;
(73)         case 'c':
(74)         case 'C':
(75)             cOption = 'c'; /* run as a consumer */
(76)             break;
(77)         case ':':
(78)         case '?':
(79)             printf("\nSYNOPSIS\n");
(80)             printf("%s %s\n", argv[0], option1);
(81)             printf("%s %s\n", argv[0], option2);
(82)             printf("%s\n", optdescription);
(83)             exit(1);
(84)             break;
(85)     }
(86) }
(87)
(88) if (cOption == 'u') {
(89)     publisher(); /* invoke producer */
(90) } else if (cOption == 'c') {
(91)     subscriber(); /* invoke consumer */
(92) } else {
(93)     printf("\nSYNOPSIS\n");
(94)     printf("%s %s\n", argv[0], option1);
(95)     printf("%s %s\n", argv[0], option2);
(96)     printf("%s\n", optdescription);
(97)     exit(1);
(98) }
(99) return 0;
(100) }
(101)
(102)
(103) /*
(104) * =====
(105) * Topic Publisher
(106) * This routine demonstrates how to publish to a topic.
(107) * =====
(108) */
(109) static void publisher()
(110) {
(111)     char                pBuffer[] = "This is a text message";
(112)     char                pTopicName[] = "PubSubSample";
(113)     int                 iMessagePriority = 4;
(114)     long                iTimeToLive = 0;
(115)
(116)     try {
(117)         /* Create a topic factory. */
(118)         TopicConnectionFactory* pTopicConnectionFactory =
(119)             LookupTopicConnectionFactory(MSCLIENT_DLL_NAME, "",
(120)             spHostName, susPort, 0, 0);
(121)
```

```
(122)    /* Create a topic connection. */
(123)    TopicConnection* pTopicConnection = pTopicConnectionFactory-
(124)    >createTopicConnection();
(125)
(126)    /* Set the client ID. */
(127)    pTopicConnection->setClientID("TopicPublisher");
(128)
(129)    /* Start the connection. */
(130)    pTopicConnection->start();
(131)
(132)    /* Create a topic session. */
(133)    TopicSession* pTopicSession = pTopicConnection-
(134)    >createTopicSession(true, Session::AUTO_ACKNOWLEDGE);
(135)
(136)    /* Create a topic. */
(137)    WString wsTopicName(pTopicName);
(138)    Topic* pTopic = pTopicSession->createTopic(wsTopicName);
(139)
(140)    /* Create a topic publisher. */
(141)    TopicPublisher* pTopicPublisher = pTopicSession-
(142)    >createPublisher(pTopic);
(143)
(144)    /* Create a text message. */
(145)    TextMessage* pTextMessage = pTopicSession->createTextMessage();
(146)
(147)    /* Clear the body (payload) of the message. */
(148)    pTextMessage->clearBody();
(149)
(150)    /* Copy in the text to be sent. */
(151)    pTextMessage->setText(pBuffer);
(152)
(153)    /* Set the JMSType of the message to "ASCII". */
(154)    pTextMessage->setJMSType("ASCII");
(155)
(156)    /* Publish the message. */
(157)    cout << "Sending Message: " << pBuffer << endl;
(158)    pTopicPublisher->publish(pTextMessage);
(159)
(160)    /* Commit the session. */
(161)    pTopicSession->commit();
(162)
(163)    /* Close and clean up. */
(164)    pTopicPublisher->close();
(165)    pTopicSession->close();
(166)    pTopicConnection->close();
(167)    delete(pTextMessage);
(168)    delete(pTopicPublisher);
(169)    delete(pTopicSession);
(170)    delete(pTopicConnection);
(171)    delete(pTopicConnectionFactory);
(172)    }
(173)    catch (JMSEException &e)
(174)    {
(175)        printf("JMS error: %s\n", e.getMessage());
(176)    }
(177) }
(178)
(179) /*
(180) * =====
(181) * Topic Subscriber
(182) * This routine demonstrates how to subscribe a message from
(183) * a topic.
(184) * =====
(185) */
```

```
(186) static void subscriber() {
(187)     char                szUserInput[80];
(188)     char                pTopicName[] = "eGatePubSubSample";
(189)
(190)     try {
(191)         /* create a topic connection factory*/
(192)         TopicConnectionFactory* pTopicConnectionFactory =
(193)             LookupTopicConnectionFactory(MSCLIENT_DLL_NAME, "", spHostName,
(194)             susPort, 0, 0);
(195)
(196)         /* create a topic connection */
(197)         TopicConnection* pTopicConnection = pTopicConnectionFactory-
(198)             >createTopicConnection();
(199)
(200)         /* set client ID */
(201)         pTopicConnection->setClientID("TopicSubscriber");
(202)
(203)         /* start connection */
(204)         pTopicConnection->start();
(205)
(206)         /* create a topic session */
(207)         TopicSession* pTopicSession = pTopicConnection-
(208)             >createTopicSession(true, Session::AUTO_ACKNOWLEDGE);
(209)
(210)
(211)         /* create a topic */
(212)         WString wsTopicName(pTopicName);
(213)         Topic* pTopic = pTopicSession->createTopic(wsTopicName);
(214)
(215)         /* create a subscriber */
(216)         TopicSubscriber* pTopicSubscriber = pTopicSession-
(217)             >createDurableSubscriber(pTopic, (char*)"TopicPublisher");
(218)
(219)         printf("Waiting for message ... \n");
(220)         TextMessage* pReceivedTextMessage;
(221)         do {
(222)             /* waiting for incoming messages */
(223)             Message* pReceivedMessage = pTopicSubscriber->receive();
(224)             pReceivedTextMessage = DYNAMIC_CAST(TextMessage,
(225)             pReceivedMessage);
(226)             pTopicSession->commit();
(227)             if (pReceivedTextMessage){
(228)                 WString wsJMSType = pReceivedTextMessage->getJMSType();
(229)                 WString wsText = pReceivedTextMessage->getText();
(230)                 string strText;
(231)                 strText = MsLocale::WideStringToString(wsText).c_str();
(232)                 cout << "Received Text Message " << strText << endl;
(233)             }
(234)             printf("Enter 'r' for receiving more message, 'q' for exit\n");
(235)             scanf("%s", szUserInput);
(236)         } while ( szUserInput[0] != 'q' && szUserInput[0] != 'Q' );
(237)
(238)         /* close and delete objects */
(239)         pTopicSubscriber->close();
(240)         pTopicSession->close();
(241)         pTopicConnection->close();
(242)         delete (pReceivedTextMessage);
(243)         delete (pTopicSubscriber);
(244)         delete (pTopicSession);
(245)         delete (pTopicConnection);
(246)         delete (pTopicConnectionFactory);
(247)     }
(248)     catch (JMSEException &e)
(249)     {
```

```
(250)         printf("JMS error: %s\n", e.getMessage());
(251)     }
(252) }
```

4.7.2. Queue Messaging (Sending/Receiving) Using C++

```
(1)  /* -----*
(2)  *   Sample code to demonstrate JMS Queue Messaging using C++.
(3)  *
(4)  * -----*
(5)  *
(6)  *   Disclaimer:
(7)  *
(8)  *   Copyright 2002 by SeeBeyond Technology Corporation.
(9)  *   All Rights Reserved.
(10) *
(11) *   Unless otherwise stipulated in a written agreement for this
(12) *   software, duly executed by SeeBeyond Technology Corporation,
(13) *   this software is provided as is without warranty of any kind.
(14) *   The entire risk as to the results and performance of this software
(15) *   is assumed by the user. SeeBeyond Technology Corporation disclaims
(16) *   all warranties, either express or implied, including but not
(17) *   limited, the implied warranties of merchantability, fitness for
(18) *   a particular purpose, title and non-infringement, with respect
(19) *   to this software.
(20) * -----*/
(21) #include    <ms.h>
(22) #include    <mslocale.h>
(23)
(24) #ifndef WIN32
(25) #include <unistd.h>
(26) #endif
(27) #if defined(WIN32)
(28) #include "sbyn_getopt.h"
(29) #endif
(30)
(31) #if defined(OS400)
(32) extern char *optarg;
(33) #endif
(34)
(35) #if defined(__gnu__)
(36) #include <getopt.h>
(37) #endif
(38)
(39) #define    MSCLIENT_DLL_NAME    "stc_msclient.dll"
(40)
(41) static    void sender();
(42) static    void receiver();
(43) char
(44)         optionProducer[] = "[ -u ] [ -p port ]
(45)         [ -h hostname ]";
(46) char
(47)         optionConsumer[] = "[ -c ] [ -p port ]
(48)         [ -h hostname ]";
(49) char
(50)         optdescription[] = "\t-u run as a
(51)         producer\n\t-c run as a consumer\n\t-p
(52)         port number\n\t-h hostname\n";
(53) char*
(54)         spHostName;
(55) char
(56)         localhost[] = "localhost";
(57) static unsigned short
(58)         susPort = 24053;
(59) int
(60)         iErr;
(61) char
(62)         szErrBuf[256];
(63)
(64) int main(int argc, char *argv[]) {
(65)     int
(66)         c;
(67)     char
(68)         cOption = 0;
```

```
(59)
(60)     spHostName = localhost;
(61)
(62)     while((c = getopt(argc, argv, ":p:h:P:H:cCUu")) != -1) {
(63)         switch(c) {
(64)             case 'p':
(65)             case 'P':
(66)                 susPort = atoi(optarg); /* setup the port number */
(67)                 break;
(68)             case 'h':
(69)             case 'H':
(70)                 spHostName = optarg; /* setup the hostname */
(71)                 break;
(72)             case 'U':
(73)             case 'u':
(74)                 cOption = 'u'; /* run as a producer */
(75)                 break;
(76)             case 'c':
(77)             case 'C':
(78)                 cOption = 'c'; /* run as a consumer */
(79)                 break;
(80)             case ':':
(81)             case '?':
(82)                 printf("\nSYNOPSIS\n");
(83)                 printf("%s %s\n", argv[0], optionProducer);
(84)                 printf("%s %s\n", argv[0], optionConsumer);
(85)                 printf("%s\n", optdescription);
(86)                 exit(1);
(87)                 break;
(88)             }
(89)         }
(90)
(91)         if (cOption == 'u') {
(92)             sender(); /* invoke producer */
(93)         } else if (cOption == 'c') {
(94)             receiver(); /* invoke consumer */
(95)         } else {
(96)             printf("\nSYNOPSIS\n");
(97)             printf("%s %s\n", argv[0], optionProducer);
(98)             printf("%s %s\n", argv[0], optionConsumer);
(99)             printf("%s\n", optdescription);
(100)            exit(1);
(101)        }
(102)    return 0;
(103) }
(104)
(105)
(106) /*
(107) * =====
(108) * Queue Sender
(109) * This routine demonstrates how to send message to a queue.
(110) * =====
(111) */
(112) static void sender()
(113) {
(114)     char                pQueueName[] = "P2PSample";
(115)     const int           MAX_MESSAGE_SIZE = 60;
(116)     char                pBuffer[] = "This is a text message";
(117)
(118)     try {
(119)         /* Create a queue connection factory */
(120)         QueueConnectionFactory* pQueueConnectionFactory =
(121)             LookupQueueConnectionFactory(MSCLIENT_DLL_NAME, "",
                spHostName, susPort, 0, 0);
```

```
(122)
(123)     /* Create a queue connection */
(124)     QueueConnection* pQueueConnection = pQueueConnectionFactory-
>createQueueConnection();
(125)
(126)     /* Set the client ID */
(127)     pQueueConnection->setClientID("SENDER");
(128)
(129)     /* Start the connection */
(130)     pQueueConnection->start();
(131)
(132)     /* Create a queue session */
(133)     QueueSession* pQueueSession = pQueueConnection-
>createQueueSession(true, Session::CLIENT_ACKNOWLEDGE);
(134)
(135)     /* Create a queue */
(136)     WString wsQueueName(pQueueName);
(137)     Queue* pQueue = pQueueSession->createQueue(wsQueueName);
(138)
(139)     /* Create a queue sender */
(140)     QueueSender* pQueueSender = pQueueSession->createSender(pQueue);
(141)
(142)     /* Create a text message */
(143)     TextMessage* pTextMessage = pQueueSession->createTextMessage();
(144)
(145)     /* set the mode to r/w */
(146)     pTextMessage->clearBody();
(147)
(148)     /* Set the JMSType of the message to "ASCII". */
(149)     pTextMessage->setJMSType("ASCII");
(150)
(151)     /* Copy in the text to be sent. */
(152)     pTextMessage->setText(pBuffer);
(153)
(154)     /* send out the message */
(155)     cout << "Sending Text Message: " << pBuffer << endl;
(156)     pQueueSender->send(pTextMessage);
(157)
(158)     /* session commit */
(159)     pQueueSession->commit();
(160)
(161)
(162)     /* close and delete the objects */
(163)     pQueueSender->close();
(164)     pQueueSession->close();
(165)     pQueueConnection->close();
(166)     delete(pTextMessage);
(167)     delete(pQueueSender);
(168)     delete(pQueue);
(169)     delete(pQueueSession);
(170)     delete(pQueueConnection);
(171)     delete(pQueueConnectionFactory);
(172) }
(173) catch (JMSEException &e)
(174) {
(175)     printf("JMS error: %s\n", e.getMessage());
(176) }
(177) }
(178)
(179) /*
(180) * =====
(181) * Queue Receiver
(182) * This routine demonstrates how to receive a message from a
(183) * queue.
```



```
(184) * =====
(185) */
(186) static void receiver() {
(187)     char                szUserInput[80];
(188)     char                pQueueName[] = "eGateP2PSample";
(189)
(190)     try {
(191)         /* Create a queue connection factory */
(192)         QueueConnectionFactory* pQueueConnectionFactory =
(193)             LookupQueueConnectionFactory(MSCLIENT_DLL_NAME, "",
spHostName, susPort, 0, 0);
(194)
(195)         /* Create a queue connection */
(196)         QueueConnection* pQueueConnection = pQueueConnectionFactory-
>createQueueConnection();
(197)
(198)         /* Set the client ID */
(199)         pQueueConnection->setClientID("RECEIVER");
(200)
(201)         /* Start the connection */
(202)         pQueueConnection->start();
(203)
(204)         /* Create a queue session */
(205)         QueueSession* pQueueSession = pQueueConnection-
>createQueueSession(true, Session::CLIENT_ACKNOWLEDGE);
(206)
(207)         /* Create a queue */
(208)         WString wsQueueName(pQueueName);
(209)         Queue* pQueue = pQueueSession->createQueue(wsQueueName);
(210)
(211)         /* Create a queue receiver */
(212)         QueueReceiver* pQueueReceiver = pQueueSession-
>createReceiver(pQueue);
(213)
(214)         printf("Waiting for message ... \n");
(215)         TextMessage* pReceivedTextMessage;
(216)         do {
(217)             /* waiting for incoming messages */
(218)             Message* pReceivedMessage = pQueueReceiver->receive();
(219)             pReceivedTextMessage = DYNAMIC_CAST(TextMessage,
pReceivedMessage);
(220)             if (pReceivedTextMessage){
(221)                 WString wsJMSType = pReceivedTextMessage->getJMSType();
(222)                 WString wsText = pReceivedTextMessage->getText();
(223)                 string strText;
(224)                 strText = MsLocale::WideStringToString(wsText).c_str();
(225)                 cout << "Received Text Message " << strText << endl;
(226)             }
(227)             printf("Enter 'r' for receiving more message, 'q' for
exit\n");
(228)             scanf("%s", szUserInput);
(229)         } while ( szUserInput[0] != 'q' && szUserInput[0] != 'Q' );
(230)
(231)         /* close and delete objects */
(232)         pQueueReceiver->close();
(233)         pQueueSession->close();
(234)         pQueueConnection->close();
(235)         delete (pReceivedTextMessage);
(236)         delete (pQueueReceiver);
(237)         delete (pQueueSession);
(238)         delete (pQueueConnection);
(239)         delete (pQueueConnectionFactory);
(240)     }
(241)     catch (JMSException &e)
```

```
(242)     {
(243)         printf("JMS error: %s\n", e.getMessage());
(244)     }
(245) }
```

4.7.3. Request-Reply Messaging Using C++

```
(1)  /* -----
(2)  * Sample code to demonstrate JMS Request-Reply messaging using C++
(3)  * -----
(4)  *
(5)  * Disclaimer:
(6)  *
(7)  * Copyright 2002 by SeeBeyond Technology Corporation.
(8)  * All Rights Reserved.
(9)  *
(10) * Unless otherwise stipulated in a written agreement for this
(11) * software,
(12) * duly executed by SeeBeyond Technology Corporation, this software is
(13) * provided as is without warranty of any kind. The entire risk as to
(14) * the results and performance of this software is assumed by the
(15) * user. SeeBeyond Technology Corporation disclaims all warranties,
(16) * either
(17) * express or implied, including but not limited, the implied
(18) * warranties
(19) * of merchantability, fitness for a particular purpose, title and
(20) * non-infringement, with respect to this software.
(21) * -----*/
(22)
(23) #include <ms.h>
(24) #include <mslocale.h>
(25)
(26) #ifndef WIN32
(27) #include <unistd.h>
(28) #endif
(29) #if defined(WIN32)
(30) #include "sbyn_getopt.h"
(31) #endif
(32)
(33) #if defined(OS400)
(34) extern char *optarg;
(35) #endif
(36)
(37) #if defined(__gnu__)
(38) #include <getopt.h>
(39) #endif
(40)
(41) #define MSCLIENT_DLL_NAME "stc_msclient.dll"
(42)
(43) static void requestor();
(44)
(45) char optionRequestor[] = "[ -r ] [ -p port ]
(46) [ -h hostname ]";
(47) char optdescription[] = "\t-r run as a
(48) requestor\n\t-
(49) p port number\n\t-h hostname\n";
(50) char* spHostName;
(51) char localhost[] = "localhost";
(52) static unsigned short susPort = 24053;
(53) char pQueueName[] = "QueueRequestorSample";
(54)
(55) int main(int argc, char *argv[]) {
(56)     int c;
(57)     char cOption = 0;
```

```

(58)
(59)     spHostName = localhost;
(60)
(61)     while((c = getopt(argc, argv, ":p:h:P:H:rR")) != -1) {
(62)         switch(c) {
(63)             case 'p':
(64)             case 'P':
(65)                 susPort = atoi(optarg); /* setup the port number */
(66)                 break;
(67)             case 'h':
(68)             case 'H':
(69)                 spHostName = optarg; /* setup the hostname */
(70)                 break;
(71)             case 'R':
(72)             case 'r':
(73)                 cOption = 'r'; /* run as a requestor */
(74)                 break;
(75)             case ':':
(76)             case '?':
(77)                 printf("\nSYNOPSIS\n");
(78)                 printf("%s %s\n", argv[0], optionRequestor);
(79)                 printf("%s\n", optdescription);
(80)                 exit(1);
(81)         }
(82)     }
(83)
(84)     if (cOption == 'r'){
(85)         requestor(); /* invoke requestor */
(86)     } else {
(87)         printf("\nSYNOPSIS\n");
(88)         printf("%s %s\n", argv[0], optionRequestor);
(89)         printf("%s\n", optdescription);
(90)         exit(1);
(91)     }
(92)     return 0;
(93) }
(94)
(95)
(96)
(97) /*
(98) * =====
(99) * Queue Requestor
(100) * This routine demonstrates how to do Request/Reply.
(101) * =====
(102) */
(103) void requestor(){
(104)     char pBuffer[] = "This is a text message";
(105)
(106)     try {
(107)         /* Create a queue connection factory */
(108)         QueueConnectionFactory* pQueueConnectionFactory =
(109)             LookupQueueConnectionFactory(MSCLIENT_DLL_NAME, "",
(110)             spHostName, susPort, 0, 0);
(111)
(112)         /* Create a queue connection */
(113)         QueueConnection* pQueueConnection = pQueueConnectionFactory-
(114)             >createQueueConnection();
(115)
(116)         /* Set the client ID */
(117)         pQueueConnection->setClientID("REQUESTOR");
(118)
(119)         /* Start the connection */
(120)         pQueueConnection->start();
(121)

```

```

(122)
(123)     /* Create a queue session */
(124)     QueueSession* pQueueSession = pQueueConnection-
(125)     >createQueueSession(false, Session::CLIENT_ACKNOWLEDGE);
(126)
(127)     /* Create a queue */
(128)     WString wsQueueName(pQueueName);
(129)     Queue* pQueue = pQueueSession->createQueue(wsQueueName);
(130)
(131)     /* Create a queue requestor */
(132)     QueueRequestor* pQueueRequestor = new QueueRequestor(pQueueSession,
(133)     pQueue);
(134)
(135)
(136)     /* Create a text message */
(137)     TextMessage* pTextMessage = pQueueSession->createTextMessage();
(138)
(139)     /* set the mode to r/w */
(140)     pTextMessage->clearBody();
(141)
(142)     /* Copy in the text to be sent. */
(143)     pTextMessage->setText(pBuffer);
(144)
(145)     /* Set ReplyTo destination */
(146)     pTextMessage->setJMSReplyTo(pQueue);
(147)
(148)     /* Make a request and wait for a reply */
(149)     Message* pReplyMessage = pQueueRequestor->request(pTextMessage,
(150)     100000);
(151)     TextMessage* pReplyTextMessage = DYNAMIC_CAST(TextMessage,
(152)     pReplyMessage);
(153)     if (pReplyTextMessage) {
(154)         WString wsText = pReplyTextMessage->getText();
(155)         string strText;
(156)         strText = MsLocale::WideStringToString(wsText).c_str();
(157)         cout << "Received Text Message " << strText << endl;
(158)     }
(159)     delete(pReplyTextMessage);
(160)
(161)     /* close and delete objects */
(162)     pQueueSession->close();
(163)     pQueueConnection->close();
(164)     delete(pTextMessage);
(165)     delete(pQueueRequestor);
(166)     delete(pQueue);
(167)     delete(pQueueSession);
(168)     delete(pQueueConnection);
(169)     delete(pQueueConnectionFactory);
(170)     }
(171)     catch (JMSException &e)
(172)     {
(173)         printf("JMS error: %s\n", e.getMessage());
(174)     }
(175) }
(176)

```

4.7.4. Message Selector Using C++

```

(1)     *-----
(2)     * Sample code to demonstrate JMS pub/sub messaging with selector.
(3)     *-----
(4)     *
(5)     * Disclaimer:
(6)     *

```

```
(7) * Copyright 2002 by SeeBeyond Technology Corporation.
(8) * All Rights Reserved.
(9) * Unless otherwise stipulated in a written agreement for this
(10) * software, duly executed by SeeBeyond Technology Corporation, this
(11) * software is provided as is without warranty of any kind. The
(12) * entire risk as to the results and performance of this software
(13) * is assumed by the user. SeeBeyond Technology Corporation disclaims
(14) * all warranties, either express or implied, including but not
(15) * limited, the implied warranties of merchantability, fitness for a
(16) * particular purpose, title and non-infringement, with respect to
(17) * this software.
(18) *
(19) * -----*/
(20)
(21) #include <ms.h>
(22) #include <mslocale.h>
(23)
(24) #ifndef WIN32
(25) #include <unistd.h>
(26) #endif
(27) #if defined(WIN32)
(28) #include "sbyn_getopt.h"
(29) #endif
(30)
(31) #if defined(OS400)
(32) extern char *optarg;
(33) #endif
(34)
(35) #if defined(__gnu__)
(36) #include <getopt.h>
(37) #endif
(38)
(39) #define MSCLIENT_DLL_NAME "stc_msclient.dll"
(40) char optionProducer[] = "[ -u ] [ -p port ]
(41) [ -h hostname ]";
(42) char optionConsumer[] = "[ -c ] [ -p port ]
(43) [ -h hostname ]";
(44) char optdescription[] = "\t-u run as a
(45) producer\n\t-c run as a consumer\n\t-p
(46) port number\n\t-h hostname\n";
(47) static char localhost[] = "localhost";
(48) static unsigned short susPort = 24053; /* default port number */
(49) unsigned long sulMessageSize = 16; /* default host name */
(50) static char* spHostName;
(51) static char PROP_NAME[] = "property";
(52)
(53)
(54) static void selector_publisher();
(55) static void selector_subscriber();
(56)
(57)
(58) int main(int argc, char *argv[]) {
(59)     int c;
(60)     char cOption = 0;
(61)
(62)     spHostName = localhost;
(63)
(64)     while((c = getopt(argc, argv, ":p:h:P:H:cCUu")) != -1) {
(65)         switch(c) {
(66)             case 'p':
(67)             case 'P':
(68)                 susPort = atoi(optarg); /* setup the port number */
(69)                 break;
(70)             case 'h':
```

```
(71)         case 'H':
(72)             spHostName = optarg;        /* setup the hostname */
(73)             break;
(74)         case 'U':
(75)         case 'u':
(76)             cOption = 'u';              /* run as a producer */
(77)             break;
(78)         case 'c':
(79)         case 'C':
(80)             cOption = 'c';              /* run as a consumer */
(81)             break;
(82)         case ':':
(83)         case '?':
(84)             printf("\nSYNOPSIS\n");
(85)             printf("%s %s\n", argv[0], optionProducer);
(86)             printf("%s %s\n", argv[0], optionConsumer);
(87)             printf("%s\n", optdescription);
(88)             exit(1);
(89)             break;
(90)         }
(91)     }
(92)
(93)     if (cOption == 'u'){
(94)         selector_publisher();           /* invoke producer */
(95)     } else if (cOption == 'c'){
(96)         selector_subscriber();         /* invoke consumer */
(97)     } else {
(98)         printf("\nSYNOPSIS\n");
(99)         printf("%s %s\n", argv[0], optionProducer);
(100)        printf("%s %s\n", argv[0], optionConsumer);
(101)        printf("%s\n", optdescription);
(102)        exit(1);
(103)    }
(104)    return 0;
(105) }
(106)
(107) /*
(108) * =====
(109) * Topic Publisher
(110) * This routine demonstrates how to publish to a topic.
(111) * =====
(112) */
(113) static void selector_publisher(){
(114)     int                ii;
(115)     static char        pTopicName[] = "Selector";
(116)
(117)     try {
(118)         /* Create a topic factory. */
(119)         TopicConnectionFactory* pTopicConnectionFactory =
(120)             LookupTopicConnectionFactory(MSCLIENT_DLL_NAME, "",
(121)             spHostName, susPort, 0, 0);
(122)
(123)         /* Create a topic connection. */
(124)         TopicConnection* pTopicConnection = pTopicConnectionFactory-
(125)             >createTopicConnection();
(126)
(127)         /* Set the client ID. */
(128)         pTopicConnection->setClientID("TopicPublisher");
(129)
(130)         /* Start the connection. */
(131)         pTopicConnection->start();
(132)
(133)         /* Create a topic session. */
(134)         TopicSession* pTopicSession = pTopicConnection-
```

```
(135) >createTopicSession(true, Session::AUTO_ACKNOWLEDGE);
(136)
(137) /* Create a topic. */
(138) WString wsTopicName(pTopicName);
(139) Topic* pTopic = pTopicSession->createTopic(wsTopicName);
(140)
(141) /* Create a topic publisher. */
(142) TopicPublisher* pTopicPublisher = pTopicSession-
(143) >createPublisher(pTopic);
(144)
(145)
(146) /* Set delivery mode as persistent */
(147) pTopicPublisher->setDeliveryMode(DeliveryMode::PERSISTENT);
(148)
(149)
(150) /* publish 10 messages to the topic */
(151) TextMessage* msglist[10];
(152) for(ii=0; ii<10 ;ii++){
(153)     int index;
(154)     char buf[80];
(155)
(156)     /* Create a text message. */
(157)     msglist[ii] = pTopicSession->createTextMessage();
(158)
(159)     /* Clear the body (payload) of the message. */
(160)     msglist[ii]->clearBody();
(161)     index = ii % 10;
(162)     sprintf(buf, "%d", index);
(163)
(164)     /* Set the string property */
(165)     msglist[ii]->setStringProperty(PROP_NAME, buf);
(166)
(167)     /* Copy in the text to be sent. */
(168)     msglist[ii]->setText("This is a text message");
(169)
(170)     /* Publish the message. */
(171)     pTopicPublisher->publish(msglist[ii]);
(172)     printf("... Published 1 message with property %s = %d\n",
(173)         PROP_NAME, ii);
(174) }
(175)
(176) /* Commit the session. */
(177) pTopicSession->commit();
(178)
(179)
(180) /* close and delete objects */
(181) pTopicPublisher->close();
(182) pTopicSession->close();
(183) pTopicConnection->close();
(184) for (ii = 0; ii < 10; ii++){
(185)     delete(msglist[ii]);
(186) }
(187) delete(pTopicPublisher);
(188) delete(pTopicSession);
(189) delete(pTopicConnection);
(190) delete(pTopicConnectionFactory);
(191) }
(192) catch (JMSException &e)
(193) {
(194)     printf("JMS error: %s\n", e.getMessage());
(195) }
(196) }
(197)
(198) /*
```

```
(199) * =====
(200) * Topic Subscriber
(201) * This routine demonstrates how to subscribe a message from a
(202) * topic.
(203) * =====
(204) */
(205) static void selector_subscriber(){
(206)     char                selectorString[80];
(207)     char                selectorSubscriberName[80];
(208)     int                 selector = 7;
(209)     char*               selectorName;
(210)     char                pTopicName[] = "eGateSelector";
(211)
(212)     try {
(213)         /* create a topic connection factory */
(214)         TopicConnectionFactory* pTopicConnectionFactory =
(215)             LookupTopicConnectionFactory(MSCLIENT_DLL_NAME, "",
(216)             spHostName, susPort, 0, 0);
(217)
(218)         /* Create a topic connection. */
(219)         TopicConnection* pTopicConnection = pTopicConnectionFactory-
(220)             >createTopicConnection();
(221)
(222)         /* Set the client ID. */
(223)         pTopicConnection->setClientID("Publisher");
(224)
(225)         /* Start the connection. */
(226)         pTopicConnection->start();
(227)
(228)         /* Create a topic session. */
(229)         TopicSession* pTopicSession = pTopicConnection-
(230)             >createTopicSession(true, Session::AUTO_ACKNOWLEDGE);
(231)
(232)         /* Create a topic. */
(233)         WString wsTopicName(pTopicName);
(234)         Topic* pTopic = pTopicSession->createTopic(wsTopicName);
(235)
(236)         /* Create subscriber with selector*/
(237)         sprintf(selectorString, "%s = '%d'", PROP_NAME, selector);
(238)         sprintf(selectorSubscriberName, "SelectorSubscriber%d", selector);
(239)         TopicSubscriber* pTopicSubscriber = pTopicSession-
(240)             >createDurableSubscriber(pTopic, selectorSubscriberName,
(241)             selectorString, 0);
(242)
(243)         /* Get message using selector */
(244)         selectorName = pTopicSubscriber->getMessageSelector();
(245)         printf("using selector: %s\n", selectorName);
(246)
(247)         Message* pMessage;
(248)         for (pMessage = pTopicSubscriber->receive();
(249)             pMessage != 0;
(250)             pMessage = pTopicSubscriber->receive(1000))
(251)         {
(252)             string strProperty = MsLocale::WideStringToString(pMessage-
(253)                 >getStringProperty(PROP_NAME)).c_str();
(254)             cout << "Received 1 message with " << PROP_NAME << " = " <<
(255)                 strProperty << endl;
(256)             delete(pMessage);
(257)         }
(258)
(259)         /* Session commit */
(260)         pTopicSession->commit();
(261)
```



```

(262)      /* Close and delete objects */
(263)      pTopicSubscriber->close();
(264)      pTopicSession->close();
(265)      pTopicConnection->close();
(266)      delete(pTopicSubscriber);
(267)      delete(pTopicSession);
(268)      delete(pTopicConnection);
(269)      delete(pTopicConnectionFactory);
(270)      }
(271)      catch (JMSEException &e)
(272)      {
(273)          printf("JMS error: %s\n", e.getMessage());
(274)      }
(275) }

```

4.7.5. XA Publish/Subscribe Messaging For JMS Using C++

```

(1)      *-----
(2)      * Sample code to demonstrate JMS Pub/Sub in XA.
(3)      *-----
(4)      *
(5)      * Disclaimer:
(6)      *
(7)      * Copyright 2002 by SeeBeyond Technology Corporation.
(8)      * All Rights Reserved.
(9)      * Unless otherwise stipulated in a written agreement for this
(10)     * software, duly executed by SeeBeyond Technology Corporation, this
(11)     * software is provided as is without warranty of any kind. The
(12)     * entire risk as to the results and performance of this software
(13)     * is assumed by the user. SeeBeyond Technology Corporation disclaims
(14)     * all warranties, either express or implied, including but not
(15)     * limited, the implied warranties of merchantability, fitness for a
(16)     * particular purpose, title and non-infringement, with respect to
(17)     * this software. *
(18)     *-----
(19)     #include <ms.h>
(20)     #include <msxa.h>
(21)     #include <mslocale.h>
(22)
(23)     #ifndef WIN32
(24)     #include <unistd.h>
(25)     #endif
(26)     #if defined(WIN32)
(27)     #include "sbyn_getopt.h"
(28)     #endif
(29)
(30)     #if defined(OS400)
(31)     extern char *optarg;
(32)     #endif
(33)
(34)     #if defined(__gnu__)
(35)     #include <getopt.h>
(36)     #endif
(37)
(38)     char optionProducer[] = "[ -u ] [ -p port ] [ -h
hostname ]";
(39)     char optionConsumer[] = "[ -c ] [ -p port ] [ -h
hostname ]";
(40)     char optdescription[] = "\t-u          run as a
producer\n\t-c run as a consumer\n\t-p port number\n\t-h
hostname\n";
(41)     static char localhost[] = "localhost";
(42)     static unsigned short susPort = 24053; /* default port number */
(43)     unsigned long sulMessageSize = 16; /* default host name */

```

```
(44) static char*          spHostName;
(45) static int           iNumMessages = 10;
(46) static char         szText[] = "This is a text message";
(47)
(48) static void XATopicPub();
(49) static void XATopicSub();
(50)
(51) #define MSCLIENT_DLL_NAME "stc_msclient.dll"
(52)
(53)
(54) int main(int argc, char *argv[]) {
(55)     int             c;
(56)     char            cOption = 0;
(57)
(58)     spHostName = localhost;
(59)
(60)     while((c = getopt(argc, argv, ":p:h:P:H:cCUu")) != -1) {
(61)         switch(c) {
(62)             case 'p':
(63)             case 'P':
(64)                 susPort = atoi(optarg); /* setup the port number */
(65)                 break;
(66)             case 'h':
(67)             case 'H':
(68)                 spHostName = optarg; /* setup the hostname */
(69)                 break;
(70)             case 'U':
(71)             case 'u':
(72)                 cOption = 'u'; /* run as a producer */
(73)                 break;
(74)             case 'c':
(75)             case 'C':
(76)                 cOption = 'c'; /* run as a consumer */
(77)                 break;
(78)             case ':':
(79)             case '?':
(80)                 printf("\nSYNOPSIS\n");
(81)                 printf("%s %s\n", argv[0], optionProducer);
(82)                 printf("%s %s\n", argv[0], optionConsumer);
(83)                 printf("%s\n", optdescription);
(84)                 exit(1);
(85)                 break;
(86)             }
(87)         }
(88)
(89)         if (cOption == 'u') {
(90)             XATopicPub(); /* invoke producer */
(91)         } else if (cOption == 'c') {
(92)             XATopicSub(); /* invoke consumer */
(93)         } else {
(94)             printf("\nSYNOPSIS\n");
(95)             printf("%s %s\n", argv[0], optionProducer);
(96)             printf("%s %s\n", argv[0], optionConsumer);
(97)             printf("%s\n", optdescription);
(98)             exit(1);
(99)         }
(100)     return 0;
(101) }
(102)
(103) /*
(104) * =====
(105) * Publish Message
(106) * This routine publishes iNumMessages to the topic
(107) * =====
```

```
(108) */
(109) static void PublishMessage(TopicPublisher* pPublisher, Message*
    pMessage, int iNumMessages)
(110) {
(111)     int ii;
(112)     for ( ii = 0; ii < iNumMessages; ii++){
(113)         pMessage->setIntProperty("Sequence", ii);
(114)         printf("Sending Message: Sequence number %d\n", ii);
(115)         pPublisher->publish(pMessage);
(116)     }
(117) }
(118)
(119)
(120) /*
(121) * =====
(122) * Receive Message
(123) *   This routine block on receiving message for maximum iWait
(124) *   seconds before return.
(125) * =====
(126) */
(127) static int SubscriberReceiveMessage(TopicSubscriber* pSub,
    TopicSession* pSession)
(128) {
(129)     int iMsgCount = 0;
(130)     Message* pRMsg = 0;
(131)     char szUserInput[8];
(132)     printf("Waiting for message ... \n");
(133)     do {
(134)         pRMsg = pSub->receive();
(135)         printf("Received Message %d\n", iMsgCount);
(136)         iMsgCount++;
(137)         if (iMsgCount >= iNumMessages){
(138)             printf("Enter 'r' for receiving more message, 'q' for
                exit\n");
(139)             scanf("%s", szUserInput);
(140)             pSession->commit();
(141)             iMsgCount = 0;
(142)         }
(143)
(144)     } while(szUserInput[0] != 'q' && szUserInput[0] != 'Q');
(145)     return iMsgCount;
(146) }
(147)
(148) /*
(149) * =====
(150) * Topic Publisher
(151) *   This routine demonstrates how to publish to a topic.
(152) * =====
(153) */
(154) void XATopicPub()
(155) {
(156)     char                pTopicName[] = "XAPubSubSample";
(157)     Xid *pXid;
(158)
(159)     try {
(160)         /* Create a topic factory. */
(161)         XATopicConnectionFactory* pXATopicConnectionFactory =
(162)             LookupXATopicConnectionFactory(MSCLIENT_DLL_NAME, "",
(163)             spHostName, susPort, 0, 0);
(164)
(165)         /* Create a topic connection. */
(166)         XATopicConnection* pXATopicConnection = pXATopicConnectionFactory-
(167)             >createXATopicConnection();
(168)
```

```
(169)      /* set client ID */
(170)      pXATopicConnection->setClientID("eGate{7E527692-770A-11D5-B139-
(171)      935EB6E85DBD}");
(172)
(173)      /* create XA session */
(174)      XATopicSession* pXATopicSession = pXATopicConnection-
(175)      >createXATopicSession();
(176)
(177)      /* get session */
(178)      TopicSession* pTopicSession = pXATopicSession->getTopicSession();
(179)
(180)      /* get XA resource */
(181)      XAResource* pXATopicResource = pXATopicSession->getXAResource();
(182)
(183)      /* create a Topic */
(184)      Topic* pTopic = pTopicSession->createTopic(pTopicName);
(185)
(186)      /* create a publisher */
(187)      TopicPublisher* pTopicPublisher = pTopicSession-
(188)      >createPublisher(pTopic);
(189)
(190)      /* connection start */
(191)      pXATopicConnection->start();
(192)
(193)      /* create xa id */
(194)      pXid = CreateXid(MSCLIENT_DLL_NAME);
(195)
(196)      /* associate the global transaction with the resource */
(197)      pXATopicResource->start(pXid, XAResource::TMNOFLAGS);
(198)
(199)      /* create a message */
(200)      TextMessage* pMessage = pXATopicSession->createTextMessage();
(201)
(202)      /* set mode to r/w */
(203)      pMessage->clearBody();
(204)
(205)      /* write bytes */
(206)      pMessage->setText((char*)szText);
(207)
(208)      /* publish message */
(209)      printf("Sending %d messages\n", iNumMessages);
(210)      pTopicPublisher->publish(pMessage);
(211)
(212)      /* xaEnd */
(213)      pXATopicResource->end(pXid, XAResource::TMSUCCESS);
(214)
(215)      /* =====
(216)      * Prepare-Rollback
(217)      * =====
(218)      */
(219)      /* xaPrepare */
(220)      pXATopicResource->prepare(pXid);
(221)
(222)      /* xaRollBack */
(223)      printf("Rolling back %d message\n", iNumMessages);
(224)      pXATopicResource->rollback(pXid);
(225)
(226)
(227)      /* =====
(228)      * Prepare-Commit
(229)      * =====
(230)      */
(231)
(232)      /* xa start */
```

```

(233) pXATopicResource->start(pXid, XAResource::TMNOFLAGS);
(234)
(235) /* send message */
(236) printf("Sending %d messages\n", iNumMessages);
(237) PublishMessage(pTopicPublisher, pMessage, iNumMessages);
(238)
(239) /* xaEnd */
(240) pXATopicResource->end(pXid, XAResource::TMSUCCESS);
(241)
(242) /* xaPrepare */
(243) if (XAResource::XA_OK != pXATopicResource->prepare(pXid))
(244) {
(245)     printf("ERROR: XAResourcePrepare failed\n");
(246) }
(247)
(248) /* xa commit */
(249) printf("Resource Commit...\n");
(250) pXATopicResource->commit(pXid, true);
(251)
(252) /* Close and clean up. */
(253) pTopicPublisher->close();
(254) pXATopicSession->close();
(255) pXATopicConnection->close();
(256) delete(pMessage);
(257) delete(pTopicPublisher);
(258) delete(pXid);
(259) delete(pXATopicSession);
(260) delete(pTopic);
(261) delete(pXATopicConnection);
(262) delete(pXATopicConnectionFactory);
(263) }
(264) catch (JMSEException &e)
(265) {
(266)     printf("JMS error: %s\n", e.getMessage());
(267) }
(268) }
(269)
(270) /*
(271) * =====
(272) *     Topic Subscriber
(273) * This routine demonstrates how to subscribe a message from a
(274) * topic.
(275) * =====
(276) */
(277) void XATopicSub()
(278) {
(279)     char                pTopicName[] = "eGateXAPubSubSample";
(280)     int                 iNumReceived = 0;
(281)     char                szUserInput[8];
(282)
(283)     try {
(284)         /* Create a topic factory. */
(285)         XATopicConnectionFactory* pXATopicConnectionFactory =
(286)             LookupXATopicConnectionFactory(MSCLIENT_DLL_NAME, "",
(287)             spHostName, susPort, 0, 0);
(288)
(289)         /* Create a topic connection. */
(290)         XATopicConnection* pXATopicConnection = pXATopicConnectionFactory-
(291)             >createXATopicConnection();
(292)
(293)
(294)         /* set client ID */
(295)         pXATopicConnection->setClientID("eGate{7E527692-770A-11D5-B139-
(296)             3456789}");

```

```
(297)
(298)     /* create XA session */
(299)     XATopicSession* pXATopicSession = pXATopicConnection-
(300)     >createXATopicSession();
(301)
(302)     /* get session */
(303)     TopicSession* pTopicSession = pXATopicSession->getTopicSession();
(304)
(305)     /* get XA resource */
(306)     XAResource* pXATopicResource = pXATopicSession->getXAResource();
(307)
(308)     /* create a Topic */
(309)     Topic* pTopic = pTopicSession->createTopic(pTopicName);
(310)
(311)     /* create a subscriber */
(312)     TopicSubscriber* pTopicSubscriber = pTopicSession-
(313)     >createDurableSubscriber(pTopic, (char*)"XATopicSubscriber");
(314)
(315)     /* connection start */
(316)     pXATopicConnection->start();
(317)
(318)     /* start xa resource */
(319)     Xid* pXid = CreateXid(MSCLIENT_DLL_NAME);
(320)
(321)
(322)     int iMsgCount = 0;
(323)     do {
(324)         Message* pRMsg = 0;
(325)         /* associate the global transaction with the resource */
(326)         pXATopicResource->start(pXid, XAResource::TMNOFLAGS);
(327)         while(iMsgCount < iNumMessages){
(328)             pRMsg = pTopicSubscriber->receive();
(329)             printf("Received Message %d\n", iMsgCount);
(330)             iMsgCount++;
(331)         }
(332)         /* xaEnd */
(333)         pXATopicResource->end(pXid, XAResource::TMSUCCESS);
(334)         pXATopicResource->commit(pXid, true);
(335)         printf("Enter 'r' for receiving more message, 'q' for
(336)         exit\n");
(337)         scanf("%s", szUserInput);
(338)
(339)         //pTopicSession->commit();
(340)         iMsgCount = 0;
(341)     } while(szUserInput[0] != 'q' && szUserInput[0] != 'Q');
(342)
(343)
(344)
(345)     /* close and delete objects */
(346)     pTopicSubscriber->close();
(347)     //pTopicSession->unsubscribe((char*)"TopicSubscriber");
(348)     pXATopicSession->close();
(349)     pXATopicConnection->close();
(350)
(351)     /* delete objects */
(352)     delete(pTopic);
(353)     delete(pTopicSubscriber);
(354)     delete(pXid);
(355)     delete(pXATopicSession);
(356)     delete(pXATopicConnection);
(357)     delete(pXATopicConnectionFactory);
(358)     }
(359)     catch (JMSEException &e)
(360)     {
```

```
(361)         printf("JMS error: %s\n", e.getMessage());
(362)     }
    }
```

4.8 Sample Code for Using JMS in C++

From the **Samples** directory located on the e*Gate installation CD-ROM, navigate to the **jmsapi** folder. Select the **C** folder and extract the sample files from this folder to the computer that you have installed the e*Gate API Kit on. The samples were built using Microsoft Visual Studio 6.0.

For the JMS API in C++, the following sample programs are provided on the product CD-ROM:

- [Publish/Subscribe Messaging Using C++](#) on page 175
- [Queue Messaging \(Sending/Receiving\) Using C++](#) on page 179
- [Request-Reply Messaging Using C++](#) on page 183
- [Message Selector Using C++](#) on page 186
- [XA Publish/Subscribe Messaging For JMS Using C++](#) on page 190

4.8.1. Publish/Subscribe Messaging Using C++

```
(1)  *-----*
(2)  * Sample code to demonstrate JMS Pub/Sub.
(3)  *-----*
(4)  *
(5)  * Disclaimer:
(6)  *
(7)  * Copyright 2002 by SeeBeyond Technology Corporation.
(8)  * All Rights Reserved.
(9)  * Unless otherwise stipulated in a written agreement for this
(10) * software, duly executed by SeeBeyond Technology Corporation, this
(11) * software is provided as is without warranty of any kind. The
(12) * entire risk as to the results and performance of this software
(13) * is assumed by the user. SeeBeyond Technology Corporation disclaims
(14) * all warranties, either express or implied, including but not
(15) * limited, the implied warranties of merchantability, fitness for a
(16) * particular purpose, title and non-infringement, with respect to
(17) * this software.
(18) *
(19) * -----*/
(20) #include <ms.h>
(21) #include <mslocale.h>
(22)
(23) #define MSCLIENT_DLL_NAME "stc_msclient.dll"
(24) char option1[] = "[ -u ] [ -p port ] [ -h hostname ]";
(25) char option2[] = "[ -c ] [ -p port ] [ -h hostname ]";
(26) char optdescription[] = "\t-u run as a
(27) producer\n\t-c run as a consumer\n\t-p port
(28) number\n\t-h hostname\n";
(29) static char localhost[] = "localhost";
(30) static unsigned short susPort = 24053; /* default port number */
(31) unsigned long sulMessageSize = 16; /* default host name */
```

```
(32) static char*          spHostName;
(33) static void          subscriber();
(34) static void          publisher();
(35)
(36)
(37) #ifndef WIN32
(38) #include <unistd.h>
(39) #endif
(40) #if defined(WIN32)
(41) #include "sbyn_getopt.h"
(42) #endif
(43)
(44) #if defined(OS400)
(45) extern char *optarg;
(46) #endif
(47)
(48) #if defined(__gnu__)
(49) #include <getopt.h>
(50) #endif
(51)
(52)
(53) int main(int argc, char *argv[]) {
(54)     int          c;
(55)     char          cOption = 0;
(56)
(57)     spHostName = localHost;
(58)
(59)     while((c = getopt(argc, argv, ":p:h:P:H:cCUu")) != -1) {
(60)         switch(c) {
(61)             case 'p':
(62)             case 'P':
(63)                 susPort = atoi(optarg); /* setup the port number */
(64)                 break;
(65)             case 'h':
(66)             case 'H':
(67)                 spHostName = optarg; /* setup the hostname */
(68)                 break;
(69)             case 'U':
(70)             case 'u':
(71)                 cOption = 'u'; /* run as a producer */
(72)                 break;
(73)             case 'c':
(74)             case 'C':
(75)                 cOption = 'c'; /* run as a consumer */
(76)                 break;
(77)             case ':':
(78)             case '?':
(79)                 printf("\nSYNOPSIS\n");
(80)                 printf("%s %s\n", argv[0], option1);
(81)                 printf("%s %s\n", argv[0], option2);
(82)                 printf("%s\n", optdescription);
(83)                 exit(1);
(84)                 break;
(85)         }
(86)     }
(87)
(88)     if (cOption == 'u') {
(89)         publisher(); /* invoke producer */
(90)     } else if (cOption == 'c') {
(91)         subscriber(); /* invoke consumer */
(92)     } else {
(93)         printf("\nSYNOPSIS\n");
(94)         printf("%s %s\n", argv[0], option1);
(95)         printf("%s %s\n", argv[0], option2);
```



```
(96)         printf("%s\n", optdescription);
(97)         exit(1);
(98)     }
(99)     return 0;
(100) }
(101)
(102)
(103) /*
(104)  * =====
(105)  * Topic Publisher
(106)  * This routine demonstrates how to publish to a topic.
(107)  * =====
(108) */
(109) static void publisher()
(110) {
(111)     char                pBuffer[] = "This is a text message";
(112)     char                pTopicName[] = "PubSubSample";
(113)     int                 iMessagePriority = 4;
(114)     long                iTimeToLive = 0;
(115)
(116)     try {
(117)         /* Create a topic factory. */
(118)         TopicConnectionFactory* pTopicConnectionFactory =
(119)             LookupTopicConnectionFactory(MSCLIENT_DLL_NAME, "",
(120)             spHostName, susPort, 0, 0);
(121)
(122)         /* Create a topic connection. */
(123)         TopicConnection* pTopicConnection = pTopicConnectionFactory-
(124)         >createTopicConnection();
(125)
(126)         /* Set the client ID. */
(127)         pTopicConnection->setClientID("TopicPublisher");
(128)
(129)         /* Start the connection. */
(130)         pTopicConnection->start();
(131)
(132)         /* Create a topic session. */
(133)         TopicSession* pTopicSession = pTopicConnection-
(134)         >createTopicSession(true, Session::AUTO_ACKNOWLEDGE);
(135)
(136)         /* Create a topic. */
(137)         WString wsTopicName(pTopicName);
(138)         Topic* pTopic = pTopicSession->createTopic(wsTopicName);
(139)
(140)         /* Create a topic publisher. */
(141)         TopicPublisher* pTopicPublisher = pTopicSession-
(142)         >createPublisher(pTopic);
(143)
(144)         /* Create a text message. */
(145)         TextMessage* pTextMessage = pTopicSession->createTextMessage();
(146)
(147)         /* Clear the body (payload) of the message. */
(148)         pTextMessage->clearBody();
(149)
(150)         /* Copy in the text to be sent. */
(151)         pTextMessage->setText(pBuffer);
(152)
(153)         /* Set the JMSType of the message to "ASCII". */
(154)         pTextMessage->setJMSType("ASCII");
(155)
(156)         /* Publish the message. */
(157)         cout << "Sending Message: " << pBuffer << endl;
(158)         pTopicPublisher->publish(pTextMessage);
(159)
```

```
(160)    /* Commit the session. */
(161)    pTopicSession->commit();
(162)
(163)    /* Close and clean up. */
(164)    pTopicPublisher->close();
(165)    pTopicSession->close();
(166)    pTopicConnection->close();
(167)    delete(pTextMessage);
(168)    delete(pTopicPublisher);
(169)    delete(pTopicSession);
(170)    delete(pTopicConnection);
(171)    delete(pTopicConnectionFactory);
(172)    }
(173)    catch (JMSEException &e)
(174)    {
(175)        printf("JMS error: %s\n", e.getMessage());
(176)    }
(177) }
(178)
(179) /*
(180) * =====
(181) * Topic Subscriber
(182) * This routine demonstrates how to subscribe a message from
(183) * a topic.
(184) * =====
(185) */
(186) static void subscriber() {
(187)     char                szUserInput[80];
(188)     char                pTopicName[] = "eGatePubSubSample";
(189)
(190)     try {
(191)         /* create a topic connection factory*/
(192)         TopicConnectionFactory* pTopicConnectionFactory =
(193)             LookupTopicConnectionFactory(MSCLIENT_DLL_NAME, "", spHostName,
(194)             susPort, 0, 0);
(195)
(196)         /* create a topic connection */
(197)         TopicConnection* pTopicConnection = pTopicConnectionFactory-
(198)             >createTopicConnection();
(199)
(200)         /* set client ID */
(201)         pTopicConnection->setClientID("TopicSubscriber");
(202)
(203)         /* start connection */
(204)         pTopicConnection->start();
(205)
(206)         /* create a topic session */
(207)         TopicSession* pTopicSession = pTopicConnection-
(208)             >createTopicSession(true, Session::AUTO_ACKNOWLEDGE);
(209)
(210)
(211)         /* create a topic */
(212)         WString wsTopicName(pTopicName);
(213)         Topic* pTopic = pTopicSession->createTopic(wsTopicName);
(214)
(215)         /* create a subscriber */
(216)         TopicSubscriber* pTopicSubscriber = pTopicSession-
(217)             >createDurableSubscriber(pTopic, (char*)"TopicPublisher");
(218)
(219)         printf("Waiting for message ... \n");
(220)         TextMessage* pReceivedTextMessage;
(221)         do {
(222)             /* waiting for incoming messages */
(223)             Message* pReceivedMessage = pTopicSubscriber->receive();
```

```

(224)         pReceivedTextMessage = DYNAMIC_CAST(TextMessage,
(225)         pReceivedMessage);
(226) pTopicSession->commit();
(227)         if (pReceivedTextMessage) {
(228)             WString wsJMSType = pReceivedTextMessage->getJMSType();
(229)             WString wsText = pReceivedTextMessage->getText();
(230)             string strText;
(231)             strText = MsLocale::WideStringToString(wsText).c_str();
(232)             cout << "Received Text Message " << strText << endl;
(233)         }
(234)         printf("Enter 'r' for receiving more message, 'q' for exit\n");
(235)         scanf("%s", szUserInput);
(236)     } while ( szUserInput[0] != 'q' && szUserInput[0] != 'Q' );
(237)
(238)     /* close and delete objects */
(239)     pTopicSubscriber->close();
(240)     pTopicSession->close();
(241)     pTopicConnection->close();
(242)     delete(pReceivedTextMessage);
(243)     delete(pTopicSubscriber);
(244)     delete(pTopicSession);
(245)     delete(pTopicConnection);
(246)     delete(pTopicConnectionFactory);
(247)     }
(248)     catch (JMSEException &e)
(249)     {
(250)         printf("JMS error: %s\n", e.getMessage());
(251)     }
(252) }

```

4.8.2. Queue Messaging (Sending/Receiving) Using C++

```

(1)  /* -----
(2)  *   Sample code to demonstrate JMS Queue Messaging using C++.
(3)  *
(4)  * -----
(5)  *
(6)  *   Disclaimer:
(7)  *
(8)  *   Copyright 2002 by SeeBeyond Technology Corporation.
(9)  *   All Rights Reserved.
(10) *
(11) *   Unless otherwise stipulated in a written agreement for this
(12) *   software, duly executed by SeeBeyond Technology Corporation,
(13) *   this software is provided as is without warranty of any kind.
(14) *   The entire risk as to the results and performance of this software
(15) *   is assumed by the user. SeeBeyond Technology Corporation disclaims
(16) *   all warranties, either express or implied, including but not
(17) *   limited, the implied warranties of merchantability, fitness for
(18) *   a particular purpose, title and non-infringement, with respect
(19) *   to this software.
(20) * -----*/
(21) #include    <ms.h>
(22) #include    <mslocale.h>
(23)
(24) #ifndef WIN32
(25) #include <unistd.h>
(26) #endif
(27) #if defined(WIN32)
(28) #include "sbyn_getopt.h"
(29) #endif
(30)
(31) #if defined(OS400)
(32) extern char *optarg;

```

```
(33) #endif
(34)
(35) #if defined(__gnu__)
(36) #include <getopt.h>
(37) #endif
(38)
(39) #define MSCLIENT_DLL_NAME "stc_msclient.dll"
(40)
(41) static void sender();
(42) static void receiver();
(43) char optionProducer[] = "[ -u ] [ -p port ]
(44) [ -h hostname ]";
(45) char optionConsumer[] = "[ -c ] [ -p port ]
(46) [ -h hostname ]";
(47) char optdescription[] = "\t-u run as a
(48) producer\n\t-c run as a consumer\n\t-p
(49) port number\n\t-h hostname\n";
(50) char* spHostName;
(51) char localhost[] = "localhost";
(52) static unsigned short susPort = 24053;
(53) int iErr;
(54) char szErrBuf[256];
(55)
(56) int main(int argc, char *argv[]) {
(57)     int c;
(58)     char cOption = 0;
(59)
(60)     spHostName = localhost;
(61)
(62)     while((c = getopt(argc, argv, ":p:h:P:H:cCUu")) != -1) {
(63)         switch(c) {
(64)             case 'p':
(65)             case 'P':
(66)                 susPort = atoi(optarg); /* setup the port number */
(67)                 break;
(68)             case 'h':
(69)             case 'H':
(70)                 spHostName = optarg; /* setup the hostname */
(71)                 break;
(72)             case 'U':
(73)             case 'u':
(74)                 cOption = 'u'; /* run as a producer */
(75)                 break;
(76)             case 'c':
(77)             case 'C':
(78)                 cOption = 'c'; /* run as a consumer */
(79)                 break;
(80)             case ':':
(81)             case '?':
(82)                 printf("\nSYNOPSIS\n");
(83)                 printf("%s %s\n", argv[0], optionProducer);
(84)                 printf("%s %s\n", argv[0], optionConsumer);
(85)                 printf("%s\n", optdescription);
(86)                 exit(1);
(87)                 break;
(88)             }
(89)         }
(90)
(91)         if (cOption == 'u') {
(92)             sender(); /* invoke producer */
(93)         } else if (cOption == 'c') {
(94)             receiver(); /* invoke consumer */
(95)         } else {
(96)             printf("\nSYNOPSIS\n");
```

```
(97)         printf("%s %s\n", argv[0], optionProducer);
(98)         printf("%s %s\n", argv[0], optionConsumer);
(99)         printf("%s\n", optdescription);
(100)        exit(1);
(101)    }
(102)    return 0;
(103) }
(104)
(105)
(106) /*
(107) * =====
(108) * Queue Sender
(109) * This routine demonstrates how to send message to a queue.
(110) * =====
(111) */
(112) static void sender()
(113) {
(114)     char                pQueueName[] = "P2PSample";
(115)     const int           MAX_MESSAGE_SIZE = 60;
(116)     char                pBuffer[] = "This is a text message";
(117)
(118)     try {
(119)         /* Create a queue connection factory */
(120)         QueueConnectionFactory* pQueueConnectionFactory =
(121)             LookupQueueConnectionFactory(MSCLIENT_DLL_NAME, "",
spHostName, susPort, 0, 0);
(122)
(123)         /* Create a queue connection */
(124)         QueueConnection* pQueueConnection = pQueueConnectionFactory-
>createQueueConnection();
(125)
(126)         /* Set the client ID */
(127)         pQueueConnection->setClientID("SENDER");
(128)
(129)         /* Start the connection */
(130)         pQueueConnection->start();
(131)
(132)         /* Create a queue session */
(133)         QueueSession* pQueueSession = pQueueConnection-
>createQueueSession(true, Session::CLIENT_ACKNOWLEDGE);
(134)
(135)         /* Create a queue */
(136)         WString wsQueueName(pQueueName);
(137)         Queue* pQueue = pQueueSession->createQueue(wsQueueName);
(138)
(139)         /* Create a queue sender */
(140)         QueueSender* pQueueSender = pQueueSession->createSender(pQueue);
(141)
(142)         /* Create a text message */
(143)         TextMessage* pTextMessage = pQueueSession->createTextMessage();
(144)
(145)         /* set the mode to r/w */
(146)         pTextMessage->clearBody();
(147)
(148)         /* Set the JMSType of the message to "ASCII". */
(149)         pTextMessage->setJMSType("ASCII");
(150)
(151)         /* Copy in the text to be sent. */
(152)         pTextMessage->setText(pBuffer);
(153)
(154)         /* send out the message */
(155)         cout << "Sending Text Message: " << pBuffer << endl;
(156)         pQueueSender->send(pTextMessage);
(157)
```

```
(158)     /* session commit */
(159)     pQueueSession->commit();
(160)
(161)
(162)     /* close and delete the objects */
(163)     pQueueSender->close();
(164)     pQueueSession->close();
(165)     pQueueConnection->close();
(166)     delete(pTextMessage);
(167)     delete(pQueueSender);
(168)     delete(pQueue);
(169)     delete(pQueueSession);
(170)     delete(pQueueConnection);
(171)     delete(pQueueConnectionFactory);
(172)     }
(173)     catch (JMSEException &e)
(174)     {
(175)         printf("JMS error: %s\n", e.getMessage());
(176)     }
(177) }
(178)
(179) /*
(180) * =====
(181) * Queue Receiver
(182) * This routine demonstrates how to receive a message from a
(183) * queue.
(184) * =====
(185) */
(186) static void receiver() {
(187)     char                szUserInput[80];
(188)     char                pQueueName[] = "eGateP2PSample";
(189)
(190)     try {
(191)         /* Create a queue connection factory */
(192)         QueueConnectionFactory* pQueueConnectionFactory =
(193)             LookupQueueConnectionFactory(MSCLIENT_DLL_NAME, "",
spHostName, susPort, 0, 0);
(194)
(195)         /* Create a queue connection */
(196)         QueueConnection* pQueueConnection = pQueueConnectionFactory-
>createQueueConnection();
(197)
(198)         /* Set the client ID */
(199)         pQueueConnection->setClientID("RECEIVER");
(200)
(201)         /* Start the connection */
(202)         pQueueConnection->start();
(203)
(204)         /* Create a queue session */
(205)         QueueSession* pQueueSession = pQueueConnection-
>createQueueSession(true, Session::CLIENT_ACKNOWLEDGE);
(206)
(207)         /* Create a queue */
(208)         WString wsQueueName(pQueueName);
(209)         Queue* pQueue = pQueueSession->createQueue(wsQueueName);
(210)
(211)         /* Create a queue receiver */
(212)         QueueReceiver* pQueueReceiver = pQueueSession-
>createReceiver(pQueue);
(213)
(214)         printf("Waiting for message ... \n");
(215)         TextMessage* pReceivedTextMessage;
(216)         do {
(217)             /* waiting for incoming messages */
```

```

(218)         Message* pReceivedMessage = pQueueReceiver->receive();
(219)         pReceivedTextMessage = DYNAMIC_CAST(TextMessage,
pReceivedMessage);
(220)         if (pReceivedTextMessage) {
(221)             WString wsJMSType = pReceivedTextMessage->getJMSType();
(222)             WString wsText = pReceivedTextMessage->getText();
(223)             string strText;
(224)             strText = MsLocale::WideStringToString(wsText).c_str();
(225)             cout << "Received Text Message " << strText << endl;
(226)         }
(227)         printf("Enter 'r' for receiving more message, 'q' for
exit\n");
(228)         scanf("%s", szUserInput);
(229)     } while ( szUserInput[0] != 'q' && szUserInput[0] != 'Q' );
(230)
(231)     /* close and delete objects */
(232)     pQueueReceiver->close();
(233)     pQueueSession->close();
(234)     pQueueConnection->close();
(235)     delete(pReceivedTextMessage);
(236)     delete(pQueueReceiver);
(237)     delete(pQueueSession);
(238)     delete(pQueueConnection);
(239)     delete(pQueueConnectionFactory);
(240)     }
(241)     catch (JMSEException &e)
(242)     {
(243)         printf("JMS error: %s\n", e.getMessage());
(244)     }
(245) }

```

4.8.3. Request-Reply Messaging Using C++

```

(1)  /* -----
(2)  * Sample code to demonstrate JMS Request-Reply messaging using C++
(3)  * -----
(4)  *
(5)  * Disclaimer:
(6)  *
(7)  * Copyright 2002 by SeeBeyond Technology Corporation.
(8)  * All Rights Reserved.
(9)  *
(10) * Unless otherwise stipulated in a written agreement for this
(11) * software,
(12) * duly executed by SeeBeyond Technology Corporation, this software is
(13) * provided as is without warranty of any kind. The entire risk as to
(14) * the results and performance of this software is assumed by the
(15) * user. SeeBeyond Technology Corporation disclaims all warranties,
(16) * either
(17) * express or implied, including but not limited, the implied
(18) * warranties
(19) * of merchantability, fitness for a particular purpose, title and
(20) * non-infringement, with respect to this software.
(21) * -----*/
(22)
(23) #include <ms.h>
(24) #include <mslocale.h>
(25)
(26) #ifndef WIN32
(27) #include <unistd.h>
(28) #endif
(29) #if defined(WIN32)
(30) #include "sbyn_getopt.h"
(31) #endif

```

```
(32)
(33) #if defined(OS400)
(34) extern char *optarg;
(35) #endif
(36)
(37) #if defined(__gnu__)
(38) #include <getopt.h>
(39) #endif
(40)
(41) #define                MSCLIENT_DLL_NAME        "stc_msclient.dll"
(42)
(43) static void requestor();
(44)
(45) char                    optionRequestor[] = "[ -r ] [ -p port ]
(46)                        [ -h hostname ]";
(47) char                    optdescription[] = "\t-r    run as a
(48)                        requestor\n\t-
(49)                        p port number\n\t-h  hostname\n";
(50) char*                    spHostName;
(51) char                    localhost[] = "localhost";
(52) static unsigned short   susPort = 24053;
(53) char                    pQueueName[] = "QueueRequestorSample";
(54)
(55) int main(int argc, char *argv[]) {
(56)     int                    c;
(57)     char                    cOption = 0;
(58)
(59)     spHostName = localhost;
(60)
(61)     while((c = getopt(argc, argv, ":p:h:P:H:rR")) != -1) {
(62)         switch(c) {
(63)             case 'p':
(64)             case 'P':
(65)                 susPort = atoi(optarg); /* setup the port number */
(66)                 break;
(67)             case 'h':
(68)             case 'H':
(69)                 spHostName = optarg; /* setup the hostname */
(70)                 break;
(71)             case 'R':
(72)             case 'r':
(73)                 cOption = 'r'; /* run as a requestor */
(74)                 break;
(75)             case ':':
(76)             case '?':
(77)                 printf("\nSYNOPSIS\n");
(78)                 printf("%s %s\n", argv[0], optionRequestor);
(79)                 printf("%s\n", optdescription);
(80)                 exit(1);
(81)             }
(82)     }
(83)
(84)     if (cOption == 'r') {
(85)         requestor(); /* invoke requestor */
(86)     } else {
(87)         printf("\nSYNOPSIS\n");
(88)         printf("%s %s\n", argv[0], optionRequestor);
(89)         printf("%s\n", optdescription);
(90)         exit(1);
(91)     }
(92)     return 0;
(93) }
(94)
(95)
```



```
(96)
(97) /*
(98)  * =====
(99)  * Queue Requestor
(100) * This routine demonstrates how to do Request/Reply.
(101) * =====
(102) */
(103) void requestor(){
(104)     char pBuffer[] = "This is a text message";
(105)
(106)     try {
(107)         /* Create a queue connection factory */
(108)         QueueConnectionFactory* pQueueConnectionFactory =
(109)             LookupQueueConnectionFactory(MSCLIENT_DLL_NAME, "",
(110)             spHostName, susPort, 0, 0);
(111)
(112)         /* Create a queue connection */
(113)         QueueConnection* pQueueConnection = pQueueConnectionFactory-
(114)             >createQueueConnection();
(115)
(116)         /* Set the client ID */
(117)         pQueueConnection->setClientID("REQUESTOR");
(118)
(119)         /* Start the connection */
(120)         pQueueConnection->start();
(121)
(122)
(123)         /* Create a queue session */
(124)         QueueSession* pQueueSession = pQueueConnection-
(125)             >createQueueSession(false, Session::CLIENT_ACKNOWLEDGE);
(126)
(127)         /* Create a queue */
(128)         WString wsQueueName(pQueueName);
(129)         Queue* pQueue = pQueueSession->createQueue(wsQueueName);
(130)
(131)         /* Create a queue requestor */
(132)         QueueRequestor* pQueueRequestor = new QueueRequestor(pQueueSession,
(133)             pQueue);
(134)
(135)
(136)         /* Create a text message */
(137)         TextMessage* pTextMessage = pQueueSession->createTextMessage();
(138)
(139)         /* set the mode to r/w */
(140)         pTextMessage->clearBody();
(141)
(142)         /* Copy in the text to be sent. */
(143)         pTextMessage->setText(pBuffer);
(144)
(145)         /* Set ReplyTo destination */
(146)         pTextMessage->setJMSReplyTo(pQueue);
(147)
(148)         /* Make a request and wait for a reply */
(149)         Message* pReplyMessage = pQueueRequestor->request(pTextMessage,
(150)             100000);
(151)         TextMessage* pReplyTextMessage = DYNAMIC_CAST(TextMessage,
(152)             pReplyMessage);
(153)         if (pReplyTextMessage){
(154)             WString wsText = pReplyTextMessage->getText();
(155)             string strText;
(156)             strText = MsLocale::WideStringToString(wsText).c_str();
(157)             cout << "Received Text Message " << strText << endl;
(158)         }
(159)         delete (pReplyTextMessage);
```

```

(160)
(161)     /* close and delete objects */
(162)     pQueueSession->close();
(163)     pQueueConnection->close();
(164)     delete(pTextMessage);
(165)     delete(pQueueRequestor);
(166)     delete(pQueue);
(167)     delete(pQueueSession);
(168)     delete(pQueueConnection);
(169)     delete(pQueueConnectionFactory);
(170)     }
(171)     catch (JMSEException &e)
(172)     {
(173)         printf("JMS error: %s\n", e.getMessage());
(174)     }
(175) }
(176)

```

4.8.4. Message Selector Using C++

```

(1)     *-----*
(2)     * Sample code to demonstrate JMS pub/sub messaging with selector.
(3)     *-----*
(4)     *
(5)     * Disclaimer:
(6)     *
(7)     * Copyright 2002 by SeeBeyond Technology Corporation.
(8)     * All Rights Reserved.
(9)     * Unless otherwise stipulated in a written agreement for this
(10)    * software, duly executed by SeeBeyond Technology Corporation, this
(11)    * software is provided as is without warranty of any kind. The
(12)    * entire risk as to the results and performance of this software
(13)    * is assumed by the user. SeeBeyond Technology Corporation disclaims
(14)    * all warranties, either express or implied, including but not
(15)    * limited, the implied warranties of merchantability, fitness for a
(16)    * particular purpose, title and non-infringement, with respect to
(17)    * this software.
(18)    *
(19)    *-----*/
(20)
(21) #include <ms.h>
(22) #include <mslocale.h>
(23)
(24) #ifndef WIN32
(25) #include <unistd.h>
(26) #endif
(27) #if defined(WIN32)
(28) #include "sbyn_getopt.h"
(29) #endif
(30)
(31) #if defined(OS400)
(32) extern char *optarg;
(33) #endif
(34)
(35) #if defined(__gnu__)
(36) #include <getopt.h>
(37) #endif
(38)
(39) #define MSCLIENT_DLL_NAME "stc_msclient.dll"
(40) char optionProducer[] = "[ -u ] [ -p port ]
(41) [ -h hostname ]";
(42) char optionConsumer[] = "[ -c ] [ -p port ]
(43) [ -h hostname ]";
(44) char optdescription[] = "\t-u run as a

```

```
(45)         producer\n\t-c run as a consumer\n\t-p
(46)         port number\n\t-h hostname\n";
(47) static char    localhost[] = "localhost";
(48) static unsigned short    susPort = 24053; /* default port number */
(49) unsigned long    sulMessageSize = 16; /* default host name */
(50) static char*    spHostName;
(51) static char PROP_NAME[] = "property";
(52)
(53)
(54) static void selector_publisher();
(55) static void selector_subscriber();
(56)
(57)
(58) int main(int argc, char *argv[]) {
(59)     int    c;
(60)     char    cOption = 0;
(61)
(62)     spHostName = localhost;
(63)
(64)     while((c = getopt(argc, argv, ":p:h:P:H:cCUu")) != -1) {
(65)         switch(c) {
(66)             case 'p':
(67)             case 'P':
(68)                 susPort = atoi(optarg); /* setup the port number */
(69)                 break;
(70)             case 'h':
(71)             case 'H':
(72)                 spHostName = optarg; /* setup the hostname */
(73)                 break;
(74)             case 'U':
(75)             case 'u':
(76)                 cOption = 'u'; /* run as a producer */
(77)                 break;
(78)             case 'c':
(79)             case 'C':
(80)                 cOption = 'c'; /* run as a consumer */
(81)                 break;
(82)             case ':':
(83)             case '?':
(84)                 printf("\nSYNOPSIS\n");
(85)                 printf("%s %s\n", argv[0], optionProducer);
(86)                 printf("%s %s\n", argv[0], optionConsumer);
(87)                 printf("%s\n", optdescription);
(88)                 exit(1);
(89)                 break;
(90)         }
(91)     }
(92)
(93)     if (cOption == 'u') {
(94)         selector_publisher(); /* invoke producer */
(95)     } else if (cOption == 'c') {
(96)         selector_subscriber(); /* invoke consumer */
(97)     } else {
(98)         printf("\nSYNOPSIS\n");
(99)         printf("%s %s\n", argv[0], optionProducer);
(100)        printf("%s %s\n", argv[0], optionConsumer);
(101)        printf("%s\n", optdescription);
(102)        exit(1);
(103)    }
(104)    return 0;
(105) }
(106)
(107) /*
(108) * =====
```

```
(109) * Topic Publisher
(110) * This routine demonstrates how to publish to a topic.
(111) * =====
(112) */
(113) static void selector_publisher() {
(114)     int                ii;
(115)     static char        pTopicName[] = "Selector";
(116)
(117)     try {
(118)         /* Create a topic factory. */
(119)         TopicConnectionFactory* pTopicConnectionFactory =
(120)             LookupTopicConnectionFactory(MSCLIENT_DLL_NAME, "",
(121)             spHostName, susPort, 0, 0);
(122)
(123)         /* Create a topic connection. */
(124)         TopicConnection* pTopicConnection = pTopicConnectionFactory-
(125)             >createTopicConnection();
(126)
(127)         /* Set the client ID. */
(128)         pTopicConnection->setClientID("TopicPublisher");
(129)
(130)         /* Start the connection. */
(131)         pTopicConnection->start();
(132)
(133)         /* Create a topic session. */
(134)         TopicSession* pTopicSession = pTopicConnection-
(135)             >createTopicSession(true, Session::AUTO_ACKNOWLEDGE);
(136)
(137)         /* Create a topic. */
(138)         WString wsTopicName(pTopicName);
(139)         Topic* pTopic = pTopicSession->createTopic(wsTopicName);
(140)
(141)         /* Create a topic publisher. */
(142)         TopicPublisher* pTopicPublisher = pTopicSession-
(143)             >createPublisher(pTopic);
(144)
(145)
(146)         /* Set delivery mode as persistent */
(147)         pTopicPublisher->setDeliveryMode(DeliveryMode::PERSISTENT);
(148)
(149)
(150)         /* publish 10 messages to the topic */
(151)         TextMessage* msglist[10];
(152)         for(ii=0; ii<10 ;ii++){
(153)             int index;
(154)             char buf[80];
(155)
(156)             /* Create a text message. */
(157)             msglist[ii] = pTopicSession->createTextMessage();
(158)
(159)             /* Clear the body (payload) of the message. */
(160)             msglist[ii]->clearBody();
(161)             index = ii % 10;
(162)             sprintf(buf, "%d", index);
(163)
(164)             /* Set the string property */
(165)             msglist[ii]->setStringProperty(PROP_NAME, buf);
(166)
(167)             /* Copy in the text to be sent. */
(168)             msglist[ii]->setText("This is a text message");
(169)
(170)             /* Publish the message. */
(171)             pTopicPublisher->publish(msglist[ii]);
(172)             printf("... Published 1 message with property %s = %d\n",
```

```
(173)         PROP_NAME, ii);
(174)     }
(175)
(176)     /* Commit the session. */
(177)     pTopicSession->commit();
(178)
(179)
(180)     /* close and delete objects */
(181)     pTopicPublisher->close();
(182)     pTopicSession->close();
(183)     pTopicConnection->close();
(184)     for (ii = 0; ii < 10; ii++){
(185)         delete(msglist[ii]);
(186)     }
(187)     delete(pTopicPublisher);
(188)     delete(pTopicSession);
(189)     delete(pTopicConnection);
(190)     delete(pTopicConnectionFactory);
(191)     }
(192)     catch (JMSEException &e)
(193)     {
(194)         printf("JMS error: %s\n", e.getMessage());
(195)     }
(196) }
(197)
(198) /*
(199) * =====
(200) * Topic Subscriber
(201) * This routine demonstrates how to subscribe a message from a
(202) * topic.
(203) * =====
(204) */
(205) static void selector_subscriber(){
(206)     char                selectorString[80];
(207)     char                selectorSubscriberName[80];
(208)     int                 selector = 7;
(209)     char*               selectorName;
(210)     char                pTopicName[] = "eGateSelector";
(211)
(212)     try {
(213)         /* create a topic connection factory */
(214)         TopicConnectionFactory* pTopicConnectionFactory =
(215)             LookupTopicConnectionFactory(MSCLIENT_DLL_NAME, "",
(216)             spHostName, susPort, 0, 0);
(217)
(218)         /* Create a topic connection. */
(219)         TopicConnection* pTopicConnection = pTopicConnectionFactory-
(220)             >createTopicConnection();
(221)
(222)         /* Set the client ID. */
(223)         pTopicConnection->setClientID("Publisher");
(224)
(225)         /* Start the connection. */
(226)         pTopicConnection->start();
(227)
(228)         /* Create a topic session. */
(229)         TopicSession* pTopicSession = pTopicConnection-
(230)             >createTopicSession(true, Session::AUTO_ACKNOWLEDGE);
(231)
(232)         /* Create a topic. */
(233)         WString wsTopicName(pTopicName);
(234)         Topic* pTopic = pTopicSession->createTopic(wsTopicName);
(235)
```

```

(236)     /* Create subscriber with selector*/
(237)     sprintf(selectorString, "%s = '%d'", PROP_NAME, selector);
(238)     sprintf(selectorSubscriberName, "SelectorSubscriber%d", selector);
(239)     TopicSubscriber* pTopicSubscriber = pTopicSession-
(240)     >createDurableSubscriber(pTopic, selectorSubscriberName,
(241)     selectorString, 0);
(242)
(243)     /* Get message using selector */
(244)     selectorName = pTopicSubscriber->getMessageSelector();
(245)     printf("using selector: %s\n", selectorName);
(246)
(247)     Message* pMessage;
(248)     for (pMessage = pTopicSubscriber->receive();
(249)         pMessage != 0;
(250)         pMessage = pTopicSubscriber->receive(1000))
(251)     {
(252)         string strProperty = MsLocale::WideStringToString(pMessage-
(253)         >getStringProperty(PROP_NAME)).c_str();
(254)         cout << "Received 1 message with " << PROP_NAME << " = " <<
(255)         strProperty << endl;;
(256)         delete(pMessage);
(257)     }
(258)
(259)     /* Session commit */
(260)     pTopicSession->commit();
(261)
(262)     /* Close and delete objects */
(263)     pTopicSubscriber->close();
(264)     pTopicSession->close();
(265)     pTopicConnection->close();
(266)     delete(pTopicSubscriber);
(267)     delete(pTopicSession);
(268)     delete(pTopicConnection);
(269)     delete(pTopicConnectionFactory);
(270) }
(271) catch (JMSEException &e)
(272) {
(273)     printf("JMS error: %s\n", e.getMessage());
(274) }
(275) }

```

4.8.5. XA Publish/Subscribe Messaging For JMS Using C++

```

(1)     *-----
(2)     * Sample code to demonstrate JMS Pub/Sub in XA.
(3)     *-----
(4)     *
(5)     * Disclaimer:
(6)     *
(7)     * Copyright 2002 by SeeBeyond Technology Corporation.
(8)     * All Rights Reserved.
(9)     * Unless otherwise stipulated in a written agreement for this
(10)    * software, duly executed by SeeBeyond Technology Corporation, this
(11)    * software is provided as is without warranty of any kind. The
(12)    * entire risk as to the results and performance of this software
(13)    * is assumed by the user. SeeBeyond Technology Corporation disclaims
(14)    * all warranties, either express or implied, including but not
(15)    * limited, the implied warranties of merchantability, fitness for a
(16)    * particular purpose, title and non-infringement, with respect to
(17)    * this software. *
(18)    * -----
(19)    #include <ms.h>
(20)    #include <msxa.h>
(21)    #include <mslocale.h>

```

```
(22)
(23) #ifndef WIN32
(24) #include <unistd.h>
(25) #endif
(26) #if defined(WIN32)
(27) #include "sbyn_getopt.h"
(28) #endif
(29)
(30) #if defined(OS400)
(31) extern char *optarg;
(32) #endif
(33)
(34) #if defined(__gnu__)
(35) #include <getopt.h>
(36) #endif
(37)
(38) char          optionProducer[] = "[ -u ] [ -p port ] [ -h
hostname ]";
(39) char          optionConsumer[] = "[ -c ] [ -p port ] [ -h
hostname ]";
(40) char          optdescription[] = "\t-u          run as a
producer\n\t-c run as a consumer\n\t-p port number\n\t-h
hostname\n";
(41) static char   localhost[] = "localhost";
(42) static unsigned short susPort = 24053; /* default port number */
(43) unsigned long  sulMessageSize = 16; /* default host name */
(44) static char*   spHostName;
(45) static int     iNumMessages = 10;
(46) static char    szText[] = "This is a text message";
(47)
(48) static void XATopicPub();
(49) static void XATopicSub();
(50)
(51) #define MSCLIENT_DLL_NAME "stc_msclient.dll"
(52)
(53)
(54) int main(int argc, char *argv[]) {
(55)     int          c;
(56)     char          cOption = 0;
(57)
(58)     spHostName = localhost;
(59)
(60)     while((c = getopt(argc, argv, ":p:h:P:H:cCUu")) != -1) {
(61)         switch(c) {
(62)             case 'p':
(63)             case 'P':
(64)                 susPort = atoi(optarg); /* setup the port number */
(65)                 break;
(66)             case 'h':
(67)             case 'H':
(68)                 spHostName = optarg; /* setup the hostname */
(69)                 break;
(70)             case 'U':
(71)             case 'u':
(72)                 cOption = 'u'; /* run as a producer */
(73)                 break;
(74)             case 'c':
(75)             case 'C':
(76)                 cOption = 'c'; /* run as a consumer */
(77)                 break;
(78)             case ':':
(79)             case '?':
(80)                 printf("\nSYNOPSIS\n");
(81)                 printf("%s %s\n", argv[0], optionProducer);
```

```
(82)         printf("%s %s\n", argv[0], optionConsumer);
(83)         printf("%s\n", optdescription);
(84)         exit(1);
(85)         break;
(86)     }
(87) }
(88)
(89) if (cOption == 'u'){
(90)     XATopicPub();           /* invoke producer */
(91) } else if (cOption == 'c'){
(92)     XATopicSub();           /* invoke consumer */
(93) } else {
(94)     printf("\nSYNOPSIS\n");
(95)     printf("%s %s\n", argv[0], optionProducer);
(96)     printf("%s %s\n", argv[0], optionConsumer);
(97)     printf("%s\n", optdescription);
(98)     exit(1);
(99) }
(100) return 0;
(101) }
(102)
(103) /*
(104) * =====
(105) * Publish Message
(106) * This routine publishes iNumMessages to the topic
(107) * =====
(108) */
(109) static void PublishMessage(TopicPublisher* pPublisher, Message*
pMessage, int iNumMessages)
(110) {
(111)     int ii;
(112)     for ( ii = 0; ii < iNumMessages; ii++){
(113)         pMessage->setIntProperty("Sequence", ii);
(114)         printf("Sending Message: Sequence number %d\n", ii);
(115)         pPublisher->publish(pMessage);
(116)     }
(117) }
(118)
(119)
(120) /*
(121) * =====
(122) * Receive Message
(123) * This routine block on receiving message for maximum iWait
(124) * seconds before return.
(125) * =====
(126) */
(127) static int SubscriberReceiveMessage(TopicSubscriber* pSub,
TopicSession* pSession)
(128) {
(129)     int iMsgCount = 0;
(130)     Message* pRMsg = 0;
(131)     char szUserInput[8];
(132)     printf("Waiting for message ... \n");
(133)     do {
(134)         pRMsg = pSub->receive();
(135)         printf("Received Message %d\n", iMsgCount);
(136)         iMsgCount++;
(137)         if (iMsgCount >= iNumMessages){
(138)             printf("Enter 'r' for receiving more message, 'q' for
exit\n");
(139)             scanf("%s", szUserInput);
(140)             pSession->commit();
(141)             iMsgCount = 0;
(142)         }
```



```
(143)
(144)     } while(szUserInput[0] != 'q' && szUserInput[0] != 'Q');
(145)     return iMsgCount;
(146) }
(147)
(148) /*
(149) * =====
(150) * Topic Publisher
(151) *   This routine demonstrates how to publish to a topic.
(152) * =====
(153) */
(154) void XATopicPub()
(155) {
(156)     char                               pTopicName[] = "XAPubSubSample";
(157)     Xid *pXid;
(158)
(159)     try {
(160)         /* Create a topic factory. */
(161)         XATopicConnectionFactory* pXATopicConnectionFactory =
(162)             LookupXATopicConnectionFactory(MSCLIENT_DLL_NAME, "",
(163)             spHostName, susPort, 0, 0);
(164)
(165)         /* Create a topic connection. */
(166)         XATopicConnection* pXATopicConnection = pXATopicConnectionFactory-
(167)             >createXATopicConnection();
(168)
(169)         /* set client ID */
(170)         pXATopicConnection->setClientID("eGate{7E527692-770A-11D5-B139-
(171)             935EB6E85DBD}");
(172)
(173)         /* create XA session */
(174)         XATopicSession* pXATopicSession = pXATopicConnection-
(175)             >createXATopicSession();
(176)
(177)         /* get session */
(178)         TopicSession* pTopicSession = pXATopicSession->getTopicSession();
(179)
(180)         /* get XA resource */
(181)         XAResource* pXATopicResource = pXATopicSession->getXAResource();
(182)
(183)         /* create a Topic */
(184)         Topic* pTopic = pTopicSession->createTopic(pTopicName);
(185)
(186)         /* create a publisher */
(187)         TopicPublisher* pTopicPublisher = pTopicSession-
(188)             >createPublisher(pTopic);
(189)
(190)         /* connection start */
(191)         pXATopicConnection->start();
(192)
(193)         /* create xa id */
(194)         pXid = CreateXid(MSCLIENT_DLL_NAME);
(195)
(196)         /* associate the global transaction with the resource */
(197)         pXATopicResource->start(pXid, XAResource::TMNOFLAGS);
(198)
(199)         /* create a message */
(200)         TextMessage* pMessage = pXATopicSession->createTextMessage();
(201)
(202)         /* set mode to r/w */
(203)         pMessage->clearBody();
(204)
(205)         /* write bytes */
(206)         pMessage->setText((char*)szText);
```

```
(207)
(208)     /* publish message */
(209)     printf("Sending %d messages\n", iNumMessages);
(210)     pTopicPublisher->publish(pMessage);
(211)
(212)     /* xaEnd */
(213)     pXATopicResource->end(pXid, XAResource::TMSUCCESS);
(214)
(215)     /* =====
(216)      * Prepare-Rollback
(217)      * =====
(218)      */
(219)     /* xaPrepare */
(220)     pXATopicResource->prepare(pXid);
(221)
(222)     /* xaRollBack */
(223)     printf("Rolling back %d message\n", iNumMessages);
(224)     pXATopicResource->rollback(pXid);
(225)
(226)
(227)     /* =====
(228)      * Prepare-Commit
(229)      * =====
(230)      */
(231)
(232)     /* xa start */
(233)     pXATopicResource->start(pXid, XAResource::TMNOFLAGS);
(234)
(235)     /* send message */
(236)     printf("Sending %d messages\n", iNumMessages);
(237)     PublishMessage(pTopicPublisher, pMessage, iNumMessages);
(238)
(239)     /* xaEnd */
(240)     pXATopicResource->end(pXid, XAResource::TMSUCCESS);
(241)
(242)     /* xaPrepare */
(243)     if (XAResource::XA_OK != pXATopicResource->prepare(pXid))
(244)     {
(245)         printf("ERROR: XAResourcePrepare failed\n");
(246)     }
(247)
(248)     /* xa commit */
(249)     printf("Resource Commit...\n");
(250)     pXATopicResource->commit(pXid, true);
(251)
(252)     /* Close and clean up. */
(253)     pTopicPublisher->close();
(254)     pXATopicSession->close();
(255)     pXATopicConnection->close();
(256)     delete(pMessage);
(257)     delete(pTopicPublisher);
(258)     delete(pXid);
(259)     delete(pXATopicSession);
(260)     delete(pTopic);
(261)     delete(pXATopicConnection);
(262)     delete(pXATopicConnectionFactory);
(263)     }
(264)     catch (JMSEException &e)
(265)     {
(266)         printf("JMS error: %s\n", e.getMessage());
(267)     }
(268) }
(269)
(270) /*
```

```

(271) * =====
(272) *      Topic Subscriber
(273) *      This routine demonstrates how to subscribe a message from a
(274) *      topic.
(275) * =====
(276) */
(277) void XATopicSub()
(278) {
(279)     char                pTopicName[] = "eGateXAPubSubSample";
(280)     int                 iNumReceived = 0;
(281)     char                szUserInput[8];
(282)
(283)     try {
(284)         /* Create a topic factory. */
(285)         XATopicConnectionFactory* pXATopicConnectionFactory =
(286)             LookupXATopicConnectionFactory(MSCLIENT_DLL_NAME, "",
(287)             spHostName, susPort, 0, 0);
(288)
(289)         /* Create a topic connection. */
(290)         XATopicConnection* pXATopicConnection = pXATopicConnectionFactory-
(291)             >createXATopicConnection();
(292)
(293)
(294)         /* set client ID */
(295)         pXATopicConnection->setClientID("eGate{7E527692-770A-11D5-B139-
(296)             3456789}");
(297)
(298)         /* create XA session */
(299)         XATopicSession* pXATopicSession = pXATopicConnection-
(300)             >createXATopicSession();
(301)
(302)         /* get session */
(303)         TopicSession* pTopicSession = pXATopicSession->getTopicSession();
(304)
(305)         /* get XA resource */
(306)         XAResource* pXATopicResource = pXATopicSession->getXAResource();
(307)
(308)         /* create a Topic */
(309)         Topic* pTopic = pTopicSession->createTopic(pTopicName);
(310)
(311)         /* create a subscriber */
(312)         TopicSubscriber* pTopicSubscriber = pTopicSession-
(313)             >createDurableSubscriber(pTopic, (char*)"XATopicSubscriber");
(314)
(315)         /* connection start */
(316)         pXATopicConnection->start();
(317)
(318)         /* start xa resource */
(319)         Xid* pXid = CreateXid(MSCLIENT_DLL_NAME);
(320)
(321)
(322)         int iMsgCount = 0;
(323)         do {
(324)             Message* pRMsg = 0;
(325)             /* associate the global transaction with the resource */
(326)             pXATopicResource->start(pXid, XAResource::TMNOFLAGS);
(327)             while(iMsgCount < iNumMessages){
(328)                 pRMsg = pTopicSubscriber->receive();
(329)                 printf("Received Message %d\n", iMsgCount);
(330)                 iMsgCount++;
(331)             }
(332)             /* xaEnd */
(333)             pXATopicResource->end(pXid, XAResource::TMSUCCESS);
(334)             pXATopicResource->commit(pXid, true);

```

```
(335)         printf("Enter 'r' for receiving more message, 'q' for
(336)         exit\n");
(337)         scanf("%s", szUserInput);
(338)
(339)         //pTopicSession->commit();
(340)         iMsgCount = 0;
(341)     } while(szUserInput[0] != 'q' && szUserInput[0] != 'Q');
(342)
(343)
(344)
(345)     /* close and delete objects */
(346)     pTopicSubscriber->close();
(347)     //pTopicSession->unsubscribe((char*)"TopicSubscriber");
(348)     pXATopicSession->close();
(349)     pXATopicConnection->close();
(350)
(351)     /* delete objects */
(352)     delete(pTopic);
(353)     delete(pTopicSubscriber);
(354)     delete(pXid);
(355)     delete(pXATopicSession);
(356)     delete(pXATopicConnection);
(357)     delete(pXATopicConnectionFactory);
(358)     }
(359)     catch (JMSEException &e)
(360)     {
(361)         printf("JMS error: %s\n", e.getMessage());
(362)     }
(363) }
```

Client Libraries for the JMS API

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

For Java

- [The Standard Specification for the JMS API](#) on page 197
- [Supported Java Classes for JMS](#) on page 197
- [Supported JMS Interfaces](#) on page 208

For COM+

- [The Message Service COM+ APIs](#) on page 291

For C

- [The C API for JMS](#) on page 355
- [Error Codes and Messages in the C API for JMS](#) on page 455

For C++

[The C++ API for Oracle Seebeyond JMS](#) on page 458

5.1 The Standard Specification for the JMS API

The JMS API specification defined by Sun is available as a Javadoc. It can be downloaded from:

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

5.2 Supported Java Classes for JMS

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

5.2.1. `com.seebeyond.jms.client.STCTopicRequestor`

Helper class to simplify making service requests.

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.JMSEException
```

Name	Description
message	The message text.

Throws

com.seebeyond.jms.client.JMSEException

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.JMSEException
```

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

The Close Method

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

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

5.2.2. com.seebeyond.jms.STCQueueRequestor

Helper class to simplify making service requests.

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.JMSEException
```

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.

5.2.3. class javax.jms.JMSEException

```
public class JMSEException
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 JMSEException Method

Construct a JMSEException with reason and errorCode for the exception.

```
public JMSEException(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 JMSEException Method

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

```
public JMSEException(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.

5.2.4. class javax.jms.IllegalStateException

```
public class IllegalStateException  
    extends JMSEException.
```

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 exception.

```
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)
```

5.2.5. `class javax.jms.InvalidClientIDException`

```
public class InvalidClientIDException  
    extends JMSEException.
```

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 exception.

```
public InvalidClientIDException(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

5.2.6. `class javax.jms.InvalidDestinationException`

```
public class InvalidDestinationException  
    extends JMSEException.
```

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 exception.

```
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

5.2.7. class javax.jms.InvalidSelectorException

```
public class InvalidSelectorException  
    extends JMSEException.
```

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

5.2.8. class javax.jms.JMSSecurityException

```
public class JMSSecurityException
    extends JMSEException
```

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 JMSSecurityException(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

5.2.9. class javax.jms.MessageEOFException

```
public class MessageEOFException
    extends JMSEException
```

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 MessageEOFException(java.lang.String reason,
    java.lang.String errorCode)
```

Name	Description
reason	A description of the exception

Name	Description
errorCode	A string specifying the vendor specific error code.

The MessageEOFException

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

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

Name	Description
reason	A description of the exception

5.2.10. class javax.jms.MessageFormatException

```
public class MessageFormatException
    extends JMSEException
```

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 MessageFormatException

Constructs a MessageFormatException with reason. The error code defaults to null.

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

Name	Description
reason	A description of the exception

5.2.11. class javax.jms.MessageNotReadableException

```
public class MessageNotReadableException
```

extends `JMSEException`.

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 defaults to null.

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

Name	Description
reason	A description of the exception

5.2.12. class `javax.jms.MessageNotWriteableException`

```
public class MessageNotWriteableException  
    extends JMSEException
```

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 defaults to null.

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

Name	Description
reason	A description of the exception

5.2.13. class javax.jms.ResourceAllocationException

```
public class ResourceAllocationException
    extends JMSEException
```

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 defaults to null.

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

Name	Description
reason	A description of the exception

5.2.14. class javax.jms.TransactionInProgressException

```
public class TransactionInProgressException
    extends JMSEException
```

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 defaults to null.

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

Name	Description
reason	A description of the exception

5.2.15. class javax.jms.TransactionRolledBackException

```
public class TransactionRolledBackException  
    extends JMSEException
```

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

The TransactionRolledBackException Method

Constructs a TransactionRolledBackException 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 defaults to null.

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

Name	Description
reason	A description of the exception

5.2.16. 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

5.3 Supported JMS Interfaces

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

5.3.1. 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()  
    throws JMSException
```

The getExceptionListener Method

Gets the `ExceptionListener` object for this connection.

```
public ExceptionListener getExceptionListener()
```


throws JMSEException

The getMetaData Method

Gets the metadata for this connection.

```
public ConnectionMetaData getMetaData()  
    throws JMSEException
```

The setClientID Method

Sets the client identifier for this connection.

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

Name	Description
clientID	The unique client identifier.

The setExceptionListener Method

Sets an exception listener for this connection.

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

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 JMSEException
```

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 JMSEException
```

5.3.2. 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 JMSEException
```

Name	Description
<code>transacted</code>	If true, the session is transacted.
<code>acknowledgeMode</code>	Indicates whether the consumer or the client will acknowledge any messages that it receives. This parameter is ignored if the session is transacted. Legal values are: <code>Session.AUTO_ACKNOWLEDGE</code> , <code>Session.CLIENT_ACKNOWLEDGE</code> and <code>Session.DUPS_OK_ACKNOWLEDGE</code> .

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

5.3.3. 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 JMSEException
```

Create an `XAQueueSession`.

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

`createQueueSession`

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

Create an `XAQueueSession`.

Specified by

createQueueSession in interface QueueConnection

Name	Description
transacted	ignored
acknowledgeMode	ignored.

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

5.3.4. 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 JMSEException
```

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: <ul style="list-style-type: none"> ▪ Session.AUTO_ACKNOWLEDGE ▪ Session.CLIENT_ACKNOWLEDGE ▪ Session.DUPS_OK_ACKNOWLEDGE

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

5.3.5. interface javax.jms.XATopicConnection

```
public interface XATopicConnection
    extends XAConnection, TopicConnection
```

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 JMSEException
```

Name	Description
transacted	Ignored; an XA TopicConnection is transacted by definition.
acknowledgeMode	Ignored.

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

createXATopicSession

Creates an XATopicSession.

```
public XATopicSession createXATopicSession()
    throws JMSEException
```

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

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

5.3.6. 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.naming.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 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.

5.3.7. 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 JMSEException
```

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 JMSEException 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 JMSEException
```

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 JMSEException 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.

5.3.8. 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.

5.3.9. 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 JMSEException
```

Throws JMSEException 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 JMSEException
```

Name	Description
userName	The caller's user name.
password	The caller's password

Throws JMSEException 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.

5.3.10. interface javax.jms.XATopicConnectionFactory

```
public interface XATopicConnectionFactory
    extends XAConnectionFactory, TopicConnectionFactory
```

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 JMSEException
```

Throws JMSEException 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 createXATopicConnection 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 JMSEException
```

Name	Description
userName	The caller's user name.
password	The caller's password

Throws JMSEException 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.

5.3.11. 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 JMSEException
```

Get the JMS version.

Throws `JMSEException` if some internal error occurs in JMS implementation during the metadata retrieval.

The `getJMSMajorVersion` Method

Gets the JMS major version number.

```
public int getJMSMajorVersion()  
    throws JMSEException
```

Throws `JMSEException` if some internal error occurs in JMS implementation during the metadata retrieval.

The `getJMSMinorVersion` Method

Gets the JMS minor version number.

```
public int getJMSMinorVersion()  
    throws JMSEException
```

Throws `JMSEException` if some internal error occurs in JMS implementation during the metadata retrieval.

The `getJMSProviderName` Method

Gets the JMS provider name.

```
public java.lang.String getJMSProviderName()  
    throws JMSEException
```

Throws `JMSEException` if some internal error occurs in JMS implementation during the metadata retrieval.

The `getProviderVersion` Method

Gets the JMS provider version.

```
public java.lang.String getProviderVersion()  
    throws JMSEException
```

Throws `JMSEException` if some internal error occurs in JMS implementation during the metadata retrieval.

The `getProviderMajorVersion` Method

Gets the JMS provider major version number.

```
public int getProviderMajorVersion()  
    throws JMSEException
```

Throws `JMSEException` if some internal error occurs in JMS implementation during the metadata retrieval.

The `getProviderMinorVersion` Method

Gets the JMS provider minor version number.

```
public int getProviderMinorVersion()  
    throws JMSEException
```

Throws `JMSEException` if some internal error occurs in JMS implementation during the metadata retrieval.

The `getJMSXPropertyNames` Method

Gets an enumeration of the JMSX property names.

```
public java.util.Enumeration getJMSXPropertyNames()  
    throws JMSEException
```

Throws `JMSEException` if some internal error occurs in JMS implementation during the property names retrieval.

5.3.12. `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 transport of the message to its destination. Retention of a message at the destination until its receipt is acknowledged is not guaranteed by a `PERSISTENT` delivery mode. Clients should assume that message retention policies are set administratively. Message retention policy governs the reliability of message delivery from destination to message consumer. For example, if a client's message storage space is exhausted, some messages as defined by a site specific message retention policy may be dropped.

A message is guaranteed to be delivered once-and-only-once by a JMS Provider if the delivery mode of the message is persistent and if the destination has a sufficient message retention policy.

`NON_PERSISTENT` Field

This is the lowest overhead delivery mode because it does not require that the message be logged to stable storage. The level of JMS provider failure that causes a `NON_PERSISTENT` message to be lost is not defined.

A JMS provider must deliver a `NON_PERSISTENT` message with an at-most-once guarantee. This means it may lose the message but it must not deliver it twice.

```
public static final int NON_PERSISTENT
```

PERSISTENT Field

This mode instructs the JMS provider to log the message to stable storage as part of the client's send operation. Only a hard media failure should cause a PERSISTENT message to be lost.

5.3.13. interface `javax.jms.Destination`

```
public interface Destination
```

A Destination object encapsulates a JMS IQ manager-specific address. public interface.

5.3.14. 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 JMSEException
```

Get the name of this queue. Clients that depend upon the name, are not portable.

Throws JMSEException if JMS implementation for Queue fails 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`

5.3.15. 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 JMSEException
```

Delete this temporary queue. If there are still existing senders or receivers still using it, then a `JMSEException` will be thrown.

Throws `JMSEException` if JMS implementation fails to delete a Temporary topic due to some internal error.

5.3.16. 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 JMSEException
```

Throws `JMSEException` if JMS implementation for `Topic` fails 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
```

5.3.17. interface `javax.jms.Topic`

```
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 `JMSEException` will be thrown.

```
public void delete()
    throws JMSEException
```

Throws `JMSEException` if JMS implementation fails to delete a Temporary queue due to some internal error.

5.3.18. 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 `JMSEException` 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(JMSEException exception)
```

Name	Description
exception	The JMS exception.

5.3.19. interface `javax.jms.Message`

```
public interface Message
    getMessageID
```

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 be 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 `JMSEException`.

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 can be referenced in message selectors regardless of whether they are supported by a connection. If they are not present in a message, they are treated like any other absent property.

JMSX properties that are set by provider on *send* are available to both the producer and the consumers of the message. JMSX properties that are 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; it is up to each provider to define specific values. 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. These properties 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. 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.

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**, or **+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** or **-57.9E2**, or a numeric value with a decimal such as **7.**, **-95.7**, or **+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 both `_` (underscore) and `$` (dollar sign). 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 any of the following: **NOT**, **AND**, **OR**, **BETWEEN**, **LIKE**, **IN**, or **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 expressions of a BETWEEN operation is **NULL**, then the value of the operation is **false**; if any of the expressions of a NOT BETWEEN operation is **NULL**, then the value of the operation is **true**.
- identifier [NOT] IN (*string-literal1, string-literal2, ...*) comparison operator where identifier 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 `_this` and **false** for `that`
 - ◆ 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, then 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 JMSEException.

The following message selector selects messages with a message type of car and color of blue and weight greater than 2500:

```
"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 JMSEException
```

Throws JMSEException 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 JMSEException
```

Name	Description
id	The identifier of the message.

Throws JMSEException 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 JMSEException
```

Throws JMSEException 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 JMSEException
```

Name	Description
long timestamp	Returns the messages timestamp.

Throws JMSEException 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 JMSEException
```

Throws JMSEException 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 JMSEException
```

Name	Description
correlationID	The correlation identifier value as an array of bytes.

Throws JMSEException 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 JMSEException
```

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 JMSEException 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 JMSEException
```

Throws JMSEException 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 JMSEException
```

Throws JMSEException 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 JMSEException
```

Name	Description
replyTo	The destination to send the response to for this message.

Throws JMSEException 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 JMSEException
```

Throws JMSEException 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 JMSEException
```

Name	Description
destination	The destination for this message.

Throws JMSEException 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 JMSEException
```

Throws JMSEException 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 JMSEException
```

Name	Description
deliveryMode	The delivery mode for this message.

Throws JMSEException 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 JMSEException
```

Throws JMSEException 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 JMSEException
```

Name	Description
redelivered	An indication of whether this message is being redelivered.

Throws JMSEException 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 JMSEException
```

The setJMSType Method

Sets the message type.

```
public void setJMSType(java.lang.String type)  
    throws JMSEException
```

Name	Description
type	The message type.

Throws JMSEException 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 JMSEException
```

Name	Description
type	The class of the message.

Throws JMSEException 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 JMSEException
```

Throws JMSEException 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 JMSEException
```

Name	Description
expiration	The messages expiration time.

Throws JMSEException 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 JMSEException
```

Throws `JMSEException` 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 JMSEException
```

Name	Description
priority	The default priority of this message.

Throws `JMSEException` 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 JMSEException
```

Throws `JMSEException` 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 JMSEException
```

Name	Description
name	The name of the property to test.

Throws `JMSEException` 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 JMSEException
```

Name	Description
name	The name of the boolean property.

Throws `JMSEException` if JMS fails to get Property due to some internal JMS error.

Throws `MessageFormatException` - if this type conversion is invalid.

The `getBytesProperty` Method

Returns the value of the byte property with the specified name.

```
public byte getBytesProperty(java.lang.String name)
    throws JMSEException
```

Name	Description
name	The name of the byte property.

Throws `JMSEException` 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 JMSEException
```

Name	Description
name	The name of the short property.

Throws `JMSEException` 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 JMSEException
```

Name	Description
name	The name of the int property.

Throws `JMSEException` 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 JMSEException
```

Name	Description
name	The name of the long property.

Throws JMSEException 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 JMSEException
```

Name	Description
name	The name of the float property.

Throws JMSEException 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 JMSEException
```

Name	Description
name	The name of the double property.

Throws JMSEException 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 JMSEException
```

Name	Description
name	The name of the String property.

Throws JMSEException 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 JMSEException
```

Name	Description
name	The name of the Java object property.

Throws JMSEException 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 JMSEException
```

Throws JMSEException 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 JMSEException
```

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

Throws JMSEException 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 JMSEException
```

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

Throws JMSEException 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 JMSEException
```

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

Throws JMSEException 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
    JMSEException
```

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

Throws JMSEException 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 JMSEException
```

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

Throws `JMSEException` 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 JMSEException
```

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

Throws `JMSEException` 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 JMSEException
```

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

Throws `JMSEException` 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 JMSEException
```

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

Throws `JMSEException` 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 so forth) and Strings.*

```
public void setObjectProperty(java.lang.String name, java.lang.Object
value)
    throws JMSEException
```

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

Throws JMSEException 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 JMSEException
```

Throws JMSEException 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 JMSEException
```

Throws JMSEException if JMS fails to due to some internal JMS error.

5.3.20.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 JMSEException
```

Throws MessageNotReadableException if message in write-only mode.

Throws JMSEException 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 JMSEException
```

Throws MessageNotReadableException if message in write-only mode.

Throws MessageEOFException if end of message stream

Throws JMSEException 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 JMSEException
```

Throws MessageNotReadableException if message in write-only mode.

Throws MessageEOFException if end of message stream

Throws JMSEException 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 JMSEException
```

Throws `MessageNotReadableException` if message in write-only mode.

Throws `MessageEOFException` if end of message stream

Throws `JMSEException` 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 JMSEException
```

Throws `MessageNotReadableException` if message in write-only mode.

Throws `MessageEOFException` if end of message stream

Throws `JMSEException` 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 JMSEException
```

Throws `MessageNotReadableException` if message in write-only mode.

Throws `MessageEOFException` if the end of the message stream is encountered.

Throws `JMSEException` 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 JMSEException
```

Throws `MessageNotReadableException` if message in write-only mode.

Throws `MessageEOFException` if end of message stream

Throws `JMSEException` 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 JMSEException
```

Throws `MessageNotReadableException` if message in write-only mode.

Throws `MessageEOFException` if end of message stream

Throws `JMSEException` 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 JMSEException
```

Throws `MessageNotReadableException` if message in write-only mode.

Throws `MessageEOFException` if end of message stream

Throws `JMSEException` 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 JMSEException
```

Throws `MessageNotReadableException` if message in write-only mode.

Throws `MessageEOFException` if the end of the message stream is encountered.

Throws `JMSEException` 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 JMSEException
```

Throws `MessageNotReadableException` if message in write-only mode.

Throws `MessageEOFException` if the end of the message stream is encountered.

Throws `JMSEException` 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 JMSEException
```

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 `MessageNotWritableException` 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 JMSEException
```

Name	Description
value	The byte value to be written.

Throws MessageNotWriteableException if message in read-only mode.

Throws JMSEException 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 JMSEException
```

Name	Description
value	The short to be written.

Throws MessageNotWriteableException if message in read-only mode.

Throws JMSEException 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 JMSEException
```

Name	Description
value	The char value to be written.

Throws MessageNotWriteableException if message in read-only mode.

Throws JMSEException 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 JMSEException
```

Name	Description
value	The int to be written.

Throws `MessageNotWriteableException` if message in read-only mode.

Throws `JMSEException` 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 JMSEException
```

Name	Description
value	The long to be written.

Throws `MessageNotWriteableException` if message in read-only mode.

Throws `JMSEException` 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 JMSEException
```

Name	Description
value	The float value to be written.

Throws `MessageNotWriteableException` if message in read-only mode.

Throws `JMSEException` 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 JMSEException
```

Name	Description
value	The double value to be written.

Throws `MessageNotWriteableException` if message in read-only mode.

Throws `JMSEException` 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 JMSEException
```

Name	Description
value	The String value to be written.

Throws `MessageNotWriteableException` if message in read-only mode.

Throws `JMSEException` 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 JMSEException
```

Name	Description
value	The byte array to be written.

Throws `MessageNotWriteableException` if message in read-only mode.

Throws `JMSEException` 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 JMSEException
```

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 `JMSEException` 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, and so forth), Strings, and byte arrays.*

```
public void writeObject(java.lang.Object value)
    throws JMSEException
```

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 JMSEException 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 JMSEException
```

Throws JMSEException if JMS fails to reset the message due to some internal JMS error.

Throws MessageFormatException if message has an invalid format

5.3.21.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("parm", 6) is equivalent to MapMessage.setObject("parm", 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 JMSEException
```

Name	Description
name	The name of the boolean.

Throws JMSEException 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 JMSEException
```

Name	Description
name	The name of the byte.

Throws JMSEException 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 JMSEException
```

Name	Description
name	The name of the short.

Throws JMSEException 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 JMSEException
```

Name	Description
name	The name of the Unicode character.

Throws JMSEException 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 JMSEException
```

Name	Description
name	The name of the int.

Throws JMSEException 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 JMSEException
```

Name	Description
name	The name of the long.

Throws JMSEException 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 JMSEException
```

Name	Description
name	The name of the float.

Throws JMSEException 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 JMSEException
```

Name	Description
name	The double value with the specified name.

Throws JMSEException 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 JMSEException
```

Name	Description
name	The name of the String.

Throws JMSEException 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 JMSEException
```

Name	Description
name	The name of the byte array.

Throws JMSEException 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 JMSEException
```

Name	Description
name	The name of the Java object.

Throws JMSEException 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 JMSEException
```

Throws JMSEException 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 JMSEException
```

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

Throws JMSEException 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 JMSEException
```

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

Throws JMSEException 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 JMSEException

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

Throws JMSEException 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 JMSEException
```

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

Throws JMSEException 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 JMSEException
```

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

Throws JMSEException 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 JMSEException
```

Name	Description
name	The name of the long.

Name	Description
value	The long value to set in the Map.

Throws JMSEException 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 JMSEException
```

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

Throws JMSEException 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 JMSEException
```

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

Throws JMSEException 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 JMSEException
```

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

Throws JMSEException 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 JMSEException
```

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 `JMSEException` 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 JMSEException
```

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 `JMSEException` 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 JMSEException
```

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 objectified primitive object types (*Integer, Double, Long, and so forth*), *Strings, and byte arrays*.

Throws `JMSEException` 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 JMSEException
```

Name	Description
name	The name of the item to test.

Throws `JMSEException` if JMS fails to read message due to some internal JMS error.

5.3.22. 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 is in read-only mode. If an attempt 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 JMSEException
```

Name	Description
object	The message data.

Throws `JMSEException` if JMS fails to read message due to some internal JMS error.

Throws `MessageFormatException` 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 JMSEException
```

Throws JMSEException if JMS fails to read message due to some internal JMS error.

Throws MessageFormatException if the object serialization fails.

5.3.23.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 attempts to write a message in read-only mode, a MessageNotWritableException is thrown.

The readBoolean Method

Reads a boolean from the stream message.

```
public boolean readBoolean()  
    throws JMSEException
```

Throws JMSEException if JMS fails to read message due to some internal JMS error.

Throws MessageEOFException if an end of message stream.

Throws MessageFormatException if the object serialization fails.

Throws MessageNotReadableException if the message is in write-only mode.

The readByte Method

Reads a byte value from the stream message.

```
public byte readByte()  
    throws JMSEException
```

Throws JMSEException if JMS fails to read message due to some internal JMS error.

Throws MessageEOFException if an end of message stream.

Throws MessageFormatException if the object serialization fails.

Throws MessageNotReadableException if the message is in write-only mode.

The readShort Method

Reads a 16-bit integer from the stream message.

```
public short readShort()  
    throws JMSEException
```

Throws JMSEException if JMS fails to read message due to some internal JMS error.

Throws MessageEOFException if an end of message stream.

Throws MessageFormatException if the object serialization fails.

Throws MessageNotReadableException if the message is in write-only mode.

The readChar Method

Reads a Unicode character value from the stream message.

```
public char readChar()  
    throws JMSEException
```

Throws JMSEException if JMS fails to read message due to some internal JMS error.

Throws MessageEOFException if an end of message stream.

Throws MessageFormatException if the object serialization fails.

Throws MessageNotReadableException if the message is in write-only mode.

The readInt Method

Reads a 32-bit integer from the stream message.

```
public int readInt()  
    throws JMSEException
```

Throws JMSEException if JMS fails to read message due to some internal JMS error.

Throws MessageEOFException if an end of message stream.

Throws MessageFormatException if the object serialization fails.

Throws MessageNotReadableException if the message is in write-only mode.

The readLong Method

Reads a 64-bit integer from the stream message.

```
public long readLong()  
    throws JMSEException
```

Throws JMSEException if JMS fails to read message due to some internal JMS error.

Throws MessageEOFException if an end of message stream.

Throws MessageFormatException if the object serialization fails.

Throws MessageNotReadableException if the message is in write-only mode.

The readFloat Method

Reads a float from the stream message.

```
public float readFloat()  
    throws JMSEException
```

Throws JMSEException if JMS fails to read message due to some internal JMS error.

Throws MessageEOFException if an end of message stream.

Throws MessageFormatException if the object serialization fails.

Throws MessageNotReadableException if the message is in write-only mode.

The readDouble Method

Reads a double from the stream message.

```
public double readDouble()  
    throws JMSEException
```

Throws JMSEException if JMS fails to read message due to some internal JMS error.

Throws MessageEOFException if an end of message stream.

Throws MessageFormatException if the object serialization fails.

Throws MessageNotReadableException if the message is in write-only mode.

The readString Method

Reads a String from the stream message.

```
public java.lang.String readString()  
    throws JMSEException
```

Throws JMSEException if JMS fails to read message due to some internal JMS error.

Throws MessageEOFException if an end of message stream.

Throws MessageFormatException if the object serialization fails.

Throws MessageNotReadableException if the message is in write-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 JMSEException
```

Name	Description
value	The buffer into which the data is read.

Throws JMSEException if JMS fails to read message due to some internal JMS error.

Throws MessageEOFException if an end of message stream.

Throws MessageFormatException if the object serialization fails.

Throws MessageNotReadableException if the message is in write-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 JMSEException
```

Note: *Byte values are returned as byte[], not Byte[].*

Throws JMSEException if JMS fails to read message due to some internal JMS error.

Throws MessageEOFException if an end of message stream.

Throws MessageNotReadableException if the message is in write-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 JMSEException
```

Throws JMSEException 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 JMSEException
```

Name	Description
value	The byte value to be written.

Throws JMSEException 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 JMSEException
```

Name	Description
value	The short value to be written.

Throws JMSEException 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 JMSEException
```

Name	Description
value	The char value to be written.

Throws JMSEException 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 JMSEException
```

Name	Description
value	The int value to be written.

Throws `JMSEException` 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 JMSEException
```

Name	Description
value	The long value to be written.

Throws `JMSEException` 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 JMSEException
```

Name	Description
value	The float value to be written.

Throws `JMSEException` 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 JMSEException
```

Name	Description
value	The double value to be written.

Throws JMSEException 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 JMSEException
```

Name	Description
value	The String value to be written.

Throws JMSEException 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 JMSEException
```

Name	Description
value	The byte array to be written.

Throws JMSEException 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 portion of 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 JMSEException
```

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.

Throws `JMSEException` 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, and so forth), Strings, and byte arrays.

```
public void writeObject(java.lang.Object value)
    throws JMSEException
```

Name	Description
value	The Java object to be written.

Throws `JMSEException` 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 JMSEException
```

Throws `JMSEException` if JMS fails to read message due to some internal JMS error.

Throws `MessageFormatException` if the message has an invalid format.

5.3.24. 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 message can be then read from and written to.

Refer to [interface `javax.jms.Message`](#) on page 220

The getText Method

Gets the string containing the data associated with the message. The default value is null.

```
public java.lang.String getText()  
    throws JMSEException
```

Throws JMSEException 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 JMSEException
```

Name	Description
string	The String containing the message data.

Throws JMSEException if JMS fails to read message due to some internal JMS error.

Throws MessageNotWriteableException if the message is in read-only mode.

5.3.25. 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 messages 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 an exception.

The getMessageSelector Method

Gets this message consumer's message selector expression.

```
public java.lang.String getMessageSelector()  
    throws JMSEException
```

Throws JMSEException 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 JMSEException
```

Throws JMSEException 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 JMSEException
```

Name	Description
listener	The listener that the messages are to be delivered to.

Throws JMSEException 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 is done within a transaction, the message remains on the consumer until the transaction commits.

```
public Message receive()  
    throws JMSEException
```

Throws JMSEException 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 JMSEException
```


Name	Description
timeout	The timeout value (in milliseconds).

The receiveNoWait Method

Receives the next message if one is immediately available.

```
public Message receiveNoWait()  
    throws JMSEException
```

Throws JMSEException 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 JMSEException
```

Throws JMSEException if JMS fails to close the consumer due to some error.

5.3.26. 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 JMSEException
```

Throws JMSEException if JMS fails to get queue for this queue receiver due to some internal error.

5.3.27. 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 destination 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 create a durable `TopicSubscriber` with the same name, and a new topic and/or message selector. Changing a durable subscription is equivalent to deleting and re-creating 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 263

The `getTopic` Method

```
public Topic getTopic()  
    throws JMSEException
```

Get the topic associated with this subscriber.

Throws `JMSEException` if JMS fails to get topic for this topic subscriber due to some internal error.

The `getNoLocal` Method

```
public boolean getNoLocal()  
    throws JMSEException
```

Get the NoLocal attribute for this TopicSubscriber. The default value for this attribute is false.

Throws JMSEException if JMS fails to get noLocal attribute for this topic subscriber due to some internal error.

5.3.28. 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.

5.3.29. 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 transacted 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 JMSEException
```

Name	Description
value	Indicates if this messages identifiers are disabled.

Throws JMSEException 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 JMSEException
```

Throws JMSEException 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 JMSEException
```

Name	Description
value	Indicates if this messages timestamps are disabled.

Throws JMSEException 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 JMSEException
```

Throws `JMSEException` if JMS fails to get an indication 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 JMSEException
```

Name	Description
<code>deliveryMode</code>	The message delivery mode for this message producer; acceptable values are <code>DeliveryMode.NON_PERSISTENT</code> and <code>DeliveryMode.PERSISTENT</code> .

Throws `JMSEException` 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 JMSEException
```

Throws `JMSEException` 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 JMSEException
```

Name	Description
<code>defaultPriority</code>	The message priority for this message producer; must be a value between 0 and 9.

Throws `JMSEException` 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 JMSEException
```

Throws `JMSEException` 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 JMSEException
```

Name	Description
timeToLive	The message time to live in milliseconds; zero is unlimited.

Throws JMSEException 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 JMSEException
```

Throws JMSEException if JMS fails to get Time to Live due to some internal error.

The close Method

```
public void close()
    throws JMSEException
```

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 JMSEException if JMS fails to close the producer due to some error.

5.3.30. 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 throw 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 JMSEException

Throws JMSEException 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 JMSEException
```

Name	Description
message	The message to be sent

Throws JMSEException 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 specifying delivery mode, priority and time to live to the queue.

```
public void send(Message message, int deliveryMode, int priority,
    long timeToLive)
    throws JMSEException
```

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 JMSEException 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 QueueSender 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 JMSEException
```

Name	Description
queue	The queue to which this message should be sent.
message	The message to be sent.

Throws JMSEException 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 JMSEException
```

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 JMSEException 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.

5.3.31.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 JMSEException
```

Throws JMSEException if JMS fails to get topic for this topic publisher due to some internal error.

The publish Method

Publish a Message to the topic using the topic's default values for deliveryMode, timeToLive, and priority.

```
public void publish(Message message)  
    throws JMSEException
```

Name	Description
message	The message to publish.

Throws JMSEException 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 values for deliveryMode, priority, and timeToLive.

```
public void publish(Message message, int deliveryMode, int priority,  
    long timeToLive)  
    throws JMSEException
```

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 JMSEException 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 JMSEException
```

Name	Description
topic	The topic to which to publish the message.
message	The message to publish.

Throws `JMSEException` 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 values for `deliveryMode`, `priority`, and `timeToLive`. 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 JMSEException
```

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 `JMSEException` 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.

5.3.32. interface `java.lang.Runnable`

5.3.33. 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 one or more registered listeners 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 its 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 JMSEException
```

Throws JMSEException 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 JMSEException
```

Throws JMSEException 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 JMSEException
```

Throws JMSEException 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 JMSEException
```

Throws JMSEException 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 JMSEException
```

Name	Description
object	The object to use to initialize this message.

Throws JMSEException 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 JMSEException
```

Throws JMSEException 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 JMSEException
```

Throws JMSEException 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 JMSEException
```

Name	Description
text	The string used to initialize this message.

Throws JMSEException 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 JMSEException
```

Throws JMSEException 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 JMSEException
```

Throws JMSEException if the JMS implementation fails to commit 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 JMSEException
```

Throws JMSEException if the JMS implementation fails to rollback 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 `JMSEException.IllegalStateException`. Closing a closed session must NOT throw an exception.

```
public void close()  
    throws JMSEException
```

Throws `JMSEException` 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 `JMSEException` if JMS implementation fails to stop message delivery and restart message send due to some internal error.

Throws `IllegalStateException` if method is called by a transacted session.

```
public void recover()  
    throws JMSEException
```

The getMessageListener Method

Return the session's distinguished message listener (optional).

Throws `JMSEException` if JMS fails to get the message listener due to an internal error in JMS Provider.

```
public MessageListener getMessageListener()  
    throws JMSEException
```

Throws `JMSEException` 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 JMSEException
```

Name	Description
listener	The message listener to associate with this session.

Throws `JMSEException` 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`

5.3.34. 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 JMSEException
```

Name	Description
queueName	The name of this queue.

Throws JMSEException 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 JMSEException
```

Name	Description
queue	The name of the queue to access.

Throws JMSEException - 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 JMSEException
```

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

Throws JMSEException 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 JMSEException
```

Name	Description
queue	The name of the queue to access, or null if this is an unidentified producer.

Throws JMSEException 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 JMSEException
```

Throws JMSEException if a session fails to create a Temporary Queue due to some JMS error.

5.3.35. 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 JMSEException
```

Name	Description
topicName	The name of this topic.

Throws JMSEException 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 JMSEException
```

Name	Description
topic	The topic to which to subscribe.

Throws JMSEException 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 JMSEException
```

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 JMSEException 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 JMSEException
```

Name	Description
topic	The non-temporary topic to which to subscribe.
name	The name used to identify this subscription.

Throws JMSEException 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 JMSEException
```

Name	Description
topic	The 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 its own connection.

Throws JMSEException 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 JMSEException
```

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

Throws JMSEException 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 JMSEException if a session fails to create a temporary topic due to some JMS error.

```
public TemporaryTopic createTemporaryTopic()  
    throws JMSEException
```

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 JMSEException
```

Name	Description
name	The name used to identify this subscription

Throws `JMSEException` if JMS fails to unsubscribe to durable subscription due to some JMS error.

Throws `InvalidDestinationException` if an invalid subscription name is specified.

5.3.36. 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 JMSEException
```

Throws JMSEException if JMS fails to return the transaction mode due to internal error in JMS Provider.

Specified by getTransacted in interface Session

Throws JMSEException 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 JMSEException
```

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 JMSEException
```

Specified by rollback in interface Session

Throws TransactionInProgressException if method is called on a XASession.

5.3.37. 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 JMSEException
```

Throws JMSEException if a JMS error occurs.

5.3.38. 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).

The `getTopicSession` Method

```
public TopicSession getTopicSession()  
    throws JMSEException
```

Gets the topic session associated with this `XATopicSession`.

Throws `JMSEException` - if a JMS error occurs.

5.3.39. interface `javax.jms.XAConnection`

```
public interface XAConnection
```

`XAConnection` extends the capability of `Connection` by providing an `XASession` (optional).

5.3.40. 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 JMSEException
```

Throws `JMSEException` 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 JMSEException
```

Specified by `createQueueSession` in interface `QueueConnection`

Name	Description
<code>transacted</code>	Ignored; XA sessions are transacted by definition.
<code>acknowledgeMode</code>	Ignored.

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

5.3.41. 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 JMSEException
```

Throws `JMSEException` 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 JMSEException
```

Name	Description
<code>transacted</code>	ignored.
<code>acknowledgeMode</code>	ignored.

Specified by `createTopicSession` in interface `TopicConnection`

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

5.3.42. interface `javax.jms.XAConnectionFactory`

```
public interface XAConnectionFactory
```

The `XAConnectionFactory` interface is a base interface for the `XAQueueConnectionFactory` and `XATopicConnectionFactory` interfaces.

5.3.43. 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 JMSEException
```

Throws `JMSEException` 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 JMSEException
```

Name	Description
userName	The caller's username
password	The caller's password.

Throws `JMSEException` 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.

5.3.44. interface javax.jms.XATopicConnectionFactory

```
public interface XATopicConnectionFactory
```

An `XATopicConnectionFactory` provides the same create options as a `TopicConnectionFactory` (optional).

5.4 Unsupported Java JMS Classes

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

5.5 Unsupported Java JMS Interfaces

- interface `javax.jms.ConnectionConsumer`
- interface `javax.jms.QueueBrowser`
- interface `javax.jms.ServerSession`
- interface `javax.jms.ServerSessionPool`

5.6 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)`

5.7 The Message Service COM+ APIs

5.7.1 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

5.7.2. 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**.

The Acknowledge Method

Acknowledges the receipt of current and previous messages.

```
BytesMessage.acknowledge
```

The ClearBody Method

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

```
BytesMessage.ClearBody
```

The ClearProperties Method

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

```
BytesMessage.ClearProperties
```

The GetProperty Method

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

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

Name	Description
name	The name of the property.

The PropertyExists Method

Checks whether a value for a specific property exists.

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

Name	Description
name	The name of the property to check.

The ReadBoolean Method

Reads a Boolean value from the bytes message stream.

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

The ReadByte Method

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

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

The ReadBytes Method

Reads a portion of the bytes message stream.

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

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

The ReadChar Method

Reads a Unicode character value from the bytes message stream.

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

The ReadDouble Method

Reads a double from the bytes message stream.

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

The ReadFloat Method

Reads a float from the bytes message stream.

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

The ReadInt Method

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

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

The ReadLong Method

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

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

The ReadShort Method

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

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

The ReadUnsignedByte Method

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

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

The ReadUnsignedShort Method

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

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

The ReadUTF Method

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

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

The Reset Method

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

```
BytesMessage.Reset
```

The SetProperty Method

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

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

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

The WriteBoolean Method

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

`BytesMessage.WriteBoolean(value as Boolean)`

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

The WriteByte Method

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

`BytesMessage.WriteByte(value As Byte)`

Name	Description
value	The byte value to be written

The WriteBytes Method

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

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

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

The WriteChar Method

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

`BytesMessage.WriteChar(value As integer)`

Name	Description
value	The Char value to be written.

The WriteDouble Method

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

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

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

The WriteFloat Method

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

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

Name	Description
value	The float value to be written.

The WriteInt Method

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

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

Name	Description
value	The float value to be written.

The WriteLong Method

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

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

Name	Description
value	The WriteLong is written as a currency.

The WriteObject Method

Currently not supported

The WriteShort Method

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

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

Name	Description
value	The short that is written.

The WriteUTF Method

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

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

Name	Description
value	The String value that is written.

5.7.3. Properties of the BytesMessage Object

The CorrelationID Property

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

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

The CorrelationIDAsBytes Property

Currently not supported.

The DeliveryMode Property

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

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

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

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

The 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 where a reply to this message will be sent. Destination can be a Topic, Queue, Temporary Topic, or a Temporary Queue.

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

The Timestamp Property

The `TimeStamp` property sets or returns the message timestamp.

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

The Type Property

The `Type` property sets or returns the message type.

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

5.7.4. The Connection Object

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

The Start Method

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

```
Connection.Start
```

The Stop Method

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

```
Connection.Stop
```

5.7.5. Properties of the Connection Object

The ClientID Property

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

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

The MetaData Property

This property is not currently supported.

5.7.6. The ConnectionFactory Object

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

There are no methods currently associated with this object.

5.7.7. 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 than one Message Server is running on the same host machine and using the same port number

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

5.7.8. The Connection MetaData Object

This Object is currently not supported.

5.7.9. The MapMessage Object

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

The Acknowledge Method

Acknowledges the receipt of current and previous messages.

```
MapMessage.Acknowledge
```

The ClearBody Method

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

```
MapMessage.ClearBody
```

The ClearProperties Method

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

```
MapMessage.ClearProperties
```

The GetBoolean Method

The GetBoolean method returns the boolean value with the given name

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

Name	Description
name	The name of the Boolean property.

The GetByte Method

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

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

Name	Description
name	The name of the byte.

The GetBytes Methods

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

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

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

The GetChar Method

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

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

Name	Description
name	The name of the Unicode character.

The GetDouble Method

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

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

Name	Description
name	The name of the double property.

The GetFloat Method

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

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

Name	Description
name	The name of the float property.

The GetInt Method

The GetInt method returns the long value with the given name

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

Name	Description
name	The name of the integer.

The GetLong Method

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

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

Name	Description
name	The name of the currency property.

The GetObject Method

The GetObject method is currently not supported.

The GetProperty Method

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

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

Name	Description
name	Name of the currency property.

The GetShort Method

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

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

Name	Description
name	The name of the short currency property.

The GetString Method

Return the String value with the given name

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

Name	Description
name	The name of the String property.

The ItemExists Method

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

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

Name	Description
name	The name of the item to check.

The PropertyExists Method

The PropertyExists method checks if a property value exists.

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

Name	Description
name	The name of the property value.

The SetBoolean Method

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

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

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

The SetByte Method

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

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

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

The SetBytes Method

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

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

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

The SetChar Method

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

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

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

The SetDouble Method

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

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

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

The SetFloat Methods

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

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

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

The SetInt Method

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

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

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

The SetLong Method

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

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

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

The SetObject Method

This method is currently not supported.

The SetProperty Method

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

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

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

The SetShort Method

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

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

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

The SetString Method

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

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

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

5.7.10. Properties of the MapMessage Object

The CorrelationID Property

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

```
MapMessage.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 where a reply to this message will be sent. Destination object could be a Topic, Queue, TemporaryTopic, or a TemporaryQueue.

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

The Timestamp Property

The TimeStamp property sets or returns the message timestamp.

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

The Type Property

The Type property sets or returns the message type.

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

5.7.11. The Message Object

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

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

The Acknowledge Method

Acknowledges the receipt of current and previous messages.

```
Message.acknowledge
```

The ClearBody Method

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

```
Message.ClearBody
```

The ClearProperties Method

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

```
Message.ClearProperties
```

The GetProperty Method

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

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

Name	Description
name	The name of the property.

The PropertyExists Method

Checks whether a value for a specific property exists.

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

Name	Description
name	The name of the property to check.

The SetProperty Method

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

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

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

5.7.12. Properties of the Message Object

The CorrelationID Property

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

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

The CorrelationIDAsBytes Property

The CorrelationIDAsBytes is not currently supported.

The DeliveryMode Property

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

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

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

The Destination Property

Currently not supported.

The Expiration Property

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

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

The MessageID Property

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

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

The Priority Property

Currently not supported.

The Redelivered Property

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

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

The ReplyTo Property

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

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

The Timestamp Property

The TimeStamp property sets or returns the message timestamp.

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

The Type Property

The Type property sets or returns the message type.

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

5.7.13. The MessageConsumer Object

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

The Close Method

The Close method closes resources on behalf of a MessageConsumer. A Message 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 <code>MessageConsumer</code> .

The ReceiveNoWait Method

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

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

5.7.14. Properties of the MessageConsumer Object

The MessageListener Property

This property is currently not supported.

The MessageSelector Property

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

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

5.7.15. The MessageListener Object

This object is currently not supported.

The OnMessage Property

This function is currently not supported.

5.7.16. The MessageProducer Object

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

There are no methods associated with this object.

5.7.17. Properties of the MessageProducer Object

The DeliveryMode Property

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

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

Name	Description
<code>msPersistent</code>	This mode instructs the JMS IQ Manager to log the message to storage as part of the client's send operation.
<code>msNon_Persistent</code>	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 <code>Non_Persistent</code> message with a one-time-only delivery. If the JMS IQ Manager goes down, the message will be lost.

The DisableMessageID Property

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

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

The DisableMessageTimestamp Property

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

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

The Priority Method

Currently not supported.

The TimeToLive Method

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

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

5.7.18. The Queue Object

A Queue object encapsulates a Message Server specific queue name.

The ToString Method

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

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

5.7.19. Properties of the Queue Object

The QueueName Property

Returns the name of this queue. Read-only.

5.7.20. The QueueBrowser Object

This object is currently not supported.

5.7.21. The QueueConnection Object

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

The CreateQueueSession Method

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

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

Name	Description
Transacted	If true, session is transacted.

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

The Start Method

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

```
QueueConnection.Start
```

The Stop Method

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

```
QueueConnection.Stop
```

5.7.22. Properties of QueueConnection Object

The ClientID Property

Returns or sets client identifier for this connection.

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

The MetaData Property

Not currently supported.

5.7.23. The QueueConnectionFactory Object

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

The CreateQueueConnection Method

Create a queue connection with a default user identity.

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

5.7.24. 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 = QueueConnectionFactory
```

The PortOffset Property

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

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

5.7.25. The QueueReceiver Object

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

The Close Method

Since a Message 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
```

5.7.26. Properties of the QueueReceiver Object

The MessageListener Property

This property is not currently supported.

The MessageSelector Property

Returns this message consumer's message selector expression.

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

The Queue Property

Returns the queue associated with this queue receiver.

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

5.7.27. The QueueRequestor Object

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

The Create Method

Constructs the QueueRequestor.

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

Name	Description
session	The QueueSession.
queue	Queue name.

The Request Method

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

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

Name	Description
message	The message.

5.7.28. The QueueSender Object

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

The Send Method

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

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

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

5.7.29. Properties of the QueueSender Object

The DeliveryMode Property

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

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

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

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

The DisableMessageID Property

Returns or sets an indication of whether message IDs are disabled

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

The DisableMessageTimestamp Property

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

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

The Priority Property

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

The Queue Property

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

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

The TimeToLive Property

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

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

5.7.30. The QueueSession Object

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

The Commit Method

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

```
QueueSession.Commit
```

The CreateBrowser Method

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

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

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

The CreateBytesMessage Method

Create a BytesMessage.

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

The CreateMapMessage Method

Create a MapMessage.

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

The CreateMessage Method

Create a Message.

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

The CreateQueue Method

Create a queue identity given a Queue name.

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

Name	Description
QueueName	The name of the queue.

The CreateReceiver Method

Create a QueueReceiver to receive messages for the specified queue.


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

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

The CreateSender Method

Create a QueueSender to send messages to the specified queue.

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

Name	Description
Queue	The name of the queue.

The CreateStreamMessage Method

Create a StreamMessage.

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

The CreateTemporaryQueue Method

Create a temporary queue.

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

The CreateTextMessage Method

Create a TextMessage.

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

Name	Description
Text	The string used to initialize this message.

The Recover Method

Stops message delivery in this session, and restart sending messages with the oldest unacknowledged message.

```
QueueSession.Recover()
```

The Rollback Method

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

```
QueueSession.Rollback()
```

The Run Method

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

```
QueueSession.Run()
```

5.7.31. Properties of the QueueSender Object

The MessageListener Property

This property is not currently supported.

The Transacted Property

Returns an indication that the session is in transacted mode.

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

5.7.32. The Session Object

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

The Commit Method

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

```
Session.Commit
```

The CreateBytesMessage Method

The CreateBytesMessage method creates a BytesMessage.

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

The CreateMapMessage Method

The CreateMapMessage method creates a MapMessage.

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

The CreateMessage Method

Create a Message.

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

The CreateStreamMessage Method

Create a StreamMessage.

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

The CreateTextMessage Method

Create a TextMessage.

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

Name	Description
Text	The string used to initialize this message.

The Recover Method

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

```
Session.Recover
```

The Rollback Method

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

```
Session.Rollback
```

The Run Method

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

```
Session.Run
```

5.7.33. Properties of the Session Object

The MessageListener Property

This property is currently not supported.

The Transacted Property

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

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

5.7.34. The StreamMessage Object

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

The Acknowledge Method

Acknowledges the receipt of current and previous messages.

```
StreamMessage.Acknowledge
```

The ClearBody Method

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

```
StreamMessage.ClearBody
```

The ClearProperties Method

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

```
StreamMessage.ClearProperties
```

The GetProperty Method

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

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

Name	Description
name	The name of the property.

The PropertyExists Method

Checks whether a value for a specific property exists.

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

Name	Description
name	The name of the property to check.

The ReadBoolean Method

Reads a Boolean value from the bytes message stream.

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

The ReadByte Method

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

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

The ReadBytes Method

Reads a portion of the bytes message stream.

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

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

The ReadChar Method

Reads a Unicode character value from the bytes message stream.

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

The ReadDouble Method

Reads a double from the bytes message stream.

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

The ReadFloat Method

Reads a float from the bytes message stream.

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

The ReadInt Method

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

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

The ReadLong Method

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

```
StreamMessage.ReadLong() As Currency
```

The ReadObject Method

Currently not supported.

The ReadShort Method

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

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

The ReadString Method

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

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

The Reset Method

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

```
StreamMessage.Reset
```

The SetProperty Method

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

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

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

The WriteBoolean Method

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

`StreamMessage.WriteBoolean(value as Boolean)`

Name	Description
value	The boolean value to be written.

The WriteByte Method

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

`StreamMessage.WriteByte(value As Byte)`

Name	Description
value	The byte value to be written.

The WriteBytes Method

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

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

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

The WriteChar Method

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

`StreamMessage.WriteChar(value As Integer)`

Name	Description
value	The char value to be written.

The WriteDouble Method

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

`StreamMessage.WriteDouble(value As Double)`

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

The WriteFloat Method

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

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

Name	Description
value	The float value to be written.

The WriteInt Method

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

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

Name	Description
value	The int value to be written.

The WriteLong Method

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

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

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

The WriteObject Method

Currently not supported

The WriteShort Method

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

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

Name	Description
value	The short that is written.

The WriteString Method

Write a string to the message stream.

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


Name	Description
value	The String value that is written.

5.7.35. Properties of the StreamMessage Object

The CorrelationID Property

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

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

The CorrelationIDAsBytes Property

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

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

The DeliveryMode Property

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

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

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

The Destination Property

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

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

The Expiration Property

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

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

The MessageID Property

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

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

The Priority Property

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

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

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

The Redelivered Property

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

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

The ReplyTo Property

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

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

The Timestamp Property

The TimeStamp property sets or returns the message timestamp.

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

The Type Property

The Type property sets or returns the message type.

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

5.7.36. The TemporaryQueue Object

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

The Delete Method

The Delete method deletes the temporary queue.

```
TemporaryQueue.Delete
```

The ToString Method

The ToString method returns a printed version of the queue name

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

5.7.37. Properties of the TemporaryQueue Object

The QueueName Property

The QueueName property returns the name of this queue.

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

5.7.38. The TemporaryTopic Object

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

The Delete Method

The Delete method deletes the temporary topic.

```
TemporaryTopic.Delete
```

The ToString Method

The ToString method returns a printed version of the topic name

```
TemporaryTopic.ToString
```

5.7.39. Properties of the TemporaryTopic Object

The TopicName Property

The TopicName property returns the name of this topic.

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

5.7.40. The TextMessage Object

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

The Acknowledge Method

Acknowledges the receipt of current and previous messages.

```
TextMessage.acknowledge
```

The ClearBody Method

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

```
TextMessage.ClearBody
```

The ClearProperties Method

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

```
TextMessage.ClearProperties
```

The GetProperty Method

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

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

Name	Description
name	The name of the property.

The PropertyExists Method

Checks whether a value for a specific property exists.

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

Name	Description
name	The name of the property to check.

The SetProperty Method

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

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

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

5.7.41. Properties of the Message Object

The CorrelationID Property

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

```
TextMessage.CorrelationID = String  
String = TextMessage.CorrelationID
```

The CorrelationIDAsBytes Property

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

```
Message.CorrelationIDAsBytes = Variant  
Variant = Message.CorrelationIDAsBytes
```

The DeliveryMode Property

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

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

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

The Destination Property

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

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

The Expiration Property

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

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

The MessageID Property

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

```
TextMessage.MessageID = String  
String = TextMessage.MessageID
```

The Priority Property

The Priority property sets or returns the priority that is assigned to this message. Possible numbers are 1 - 9. (Not currently supported, but suggested that the value be entered, to prevent code changes later.)

```
TextMessage.Priority = PriorityConstant  
PriorityConstant = TextMessage.Priority
```

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

The Redelivered Property

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

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

The ReplyTo Property

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

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

The Text Property

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

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

The Timestamp Property

The TimeStamp property sets or returns the message timestamp.

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

The Type Property

The Type property sets or returns the message type.

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

5.7.42. The Topic Object

A Topic object encapsulates a Message Server specific topic name.

The ToString Method

The ToString method returns a printed version of the topic name

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

5.7.43. Properties of the Topic Object

The TopicName Property

The TopicName property returns the name of this topic.

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

5.7.44. The TopicConnection Object

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

The CreateTopicSession Method

Create a TopicSession

```
TopicConnection.CreateTopicSession(Transacted As Boolean,  
acknowledgeMode As AcknowledgeModeConstants) As TopicSession
```

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

The Start Method

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


```
TopicConnection.Start
```

The Stop Method

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

```
TopicConnection.Stop
```

5.7.45. Properties of the TopicConnection

The ClientID Property

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

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

The MetaData Property

This property is currently not supported.

5.7.46. The TopicConnectionFactory Object

A client uses a TopicConnectionFactory to create TopicConnections with a Pub/Sub Message Server.

The CreateTopicConnection Method

Create a topic connection with default user identity.

```
TopicConnectionFactory.CreateTopicConnection() As TopicConnection
```

5.7.47. Properties of the TopicConnectionFactory

The HostName Property

The HostName property sets or returns the host name of the machine that the JMS IQ Server is running on.

```
TopicConnectionFactory.HostName = String  
String = TopicConnectionFactory.HostName
```

The Port Property

The Port property sets or returns the port number that the JMS IQ Server is listening on, default value is 7555

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

The PortOffset Property

The PortOffset sets or returns the port offset number of the JMS IQ Server if more than one Message Server is running on same host machine and using same port number.

```
TopicConnectionFactory.PortOffset = Long  
Long = TopicConnectionFactory
```

5.7.48. The TopicPublisher Object

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

The Publish Method

The Publish method publishes a Message to a topic for an unidentified message producer, specifying delivery mode, priority and time to live.

```
TopicPublisher.Publish(message As message, [DeliveryMode],  
[Priority], [TimeToLive], [Topic])
```

Name	Description
message	The message to publish.
deliveryMode	The delivery mode to use.
priority	The priority for this message. While not currently supported, it is recommended to implement now, to prevent code changes later.
timeToLive	The message's lifetime (in milliseconds).
topic	The topic to publish this message to.

5.7.49. Properties of TopicPublisher

The DeliveryMode Property

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

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

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

The DisableMessageID Property

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

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

The DisableMessageTimestamp Property

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

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

The Priority Property

The Priority property sets or returns the priority that is assigned to this message. Possible numbers are 1 - 9. While not currently supported, it is suggested that the desired value be entered now, to prevent code changes later.

```
TopicPublisher.Priority = PriorityConstant
PriorityConstant = TopicPublisher.Priority
```

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

The TimeToLive Property

The TimeToLive property sets and returns the default length of time in milliseconds from its dispatch time that a produced message should be retained by the message system.

```
TopicPublisher.TimeToLive = MessageConstant  
MessageConstant = TopicPublisher.TimeToLive
```

Name	Description
msDefaultTimeToLive	The default value of 0 = Unlimited

The Topic Property

The Topic property returns the topic associated with this publisher.

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

5.7.50. The TopicRequestor Object

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

The Create Method

Constructs the TopicRequestor.

```
TopicRequestor.Create(session As TopicSession, Topic As Topic)
```

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

The Request Method

Send a request and wait for a reply

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

Name	Description
message	The message text.

5.7.51. The TopicSession Object

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

The Commit Method

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

```
TopicSession.Commit
```

The CreateBytesMessage Method

The CreateBytesMessage method creates a BytesMessage.

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

The CreateDurableSubscriber Method

The CreateDurableSubscriber method creates a durable Subscriber to the specified topic.

```
TopicSession.CreateDurableSubscriber(Topic As Topic, name As String,  
[MessageSelector], [NoLocal] As TopicSubscriber
```

Name	Description
topic	The non-temporary topic to subscribe to.
name	The name used to identify this subscription.
messageSelector	Only messages with properties matching the message selector expression are delivered. You may use a null.
noLocal	If set, noLocal inhibits the delivery of messages published by its own connection.

The CreateMapMessage Method

The CreateMapMessage method creates a MapMessage.

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

The CreateMessage Method

Create a Message.

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

The CreatePublisher Method

Create a Publisher for the specified topic.

```
TopicSession.CreatePublisher(Topic As Topic) As TopicPublisher
```

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

The CreateStreamMessage Method

Create a StreamMessage.

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

The CreateSubscriber Method

Create a non-durable Subscriber to the specified topic

```
TopicSession.CreateSubscriber(Topic As Topic, [MessageSelector],  
[NoLocal]) As TopicSubscriber
```

Name	Description
topic	The topic to subscribe to.
messageSelector	Only messages with properties matching the message selector expression are delivered. This value may be null.
noLocal	If set, inhibits the delivery of messages published by its own connection.

The CreateTemporaryTopic Method

The CreateTemporaryTopic method creates a temporary topic.

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

The CreateTextMessage Method

The CreateTextMessage method creates TextMessage.

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

Name	Description
text	The string used to initialize this message.

The CreateTopic Method

Create a topic identity given a Topic name.

```
TopicSession.CreateTopic(TopicName As String) As Topic
```

Name	Description
topicName	The name of this topic.

The Recover Method

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

```
TopicSession.Recover
```

The Rollback Method

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

```
TopicSession.Rollback
```

The Run Method

The Run method is an optional method

```
TopicSession.Run
```

The Unsubscribe Method

The Unsubscribe method unsubscribes a durable subscription that has been created by a client.

```
TopicSession.Unsubscribe(name As String)
```

Name	Description
name	The name used to identify this subscription.

5.7.52. Properties of the TopicSession Object

The MessageListener Property

This property is currently not supported.

The Transacted Property

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

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

5.7.53. The TopicSubscriber Object

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

The Close Method

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

```
TopicSubscriber.Close
```

The Receive Method

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

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

Name	Description
timeout	The timeout value (in milliseconds).

The ReceiveNoWait Method

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

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

5.7.54. Properties of the TopicSubscriber Object

The MessageListener Property

This property is currently not supported.

The MessageSelector Property

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

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

The NoLocal Property

The NoLocal property returns the NoLocal attribute for this TopicSubscriber.

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

The Topic Property

The Topic property returns the topic associated with this subscriber.


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

5.7.55. The XAQueueConnection Object

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

The CreateQueueSession Method

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

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

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

The CreateXAQueueSession Method

Create an XAQueueSession.

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

The Start Method

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

```
XAQueueConnection.Start
```

The Stop Method

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

```
XAQueueConnection.Stop
```

5.7.56. Properties of XAQueueConnection Object

The ClientID Property

Returns or sets client identifier for this connection.

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

The MetaData Property

Not currently supported.

5.7.57. The XAQueueConnectionFactory Object

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

The CreateQueueConnection Method

Create a queue connection with a default user identity.

```
XAQueueConnectionFactory.CreateQueueConnection() As QueueConnection
```

The CreateXAQueueConnection Method

Create an XA queue connection with a default user identity.

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

5.7.58. 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 = XAQueueConnectionFactory
```

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
```

5.7.59. The XAQueueSession Object

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

The Commit Method

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

```
XAQueueSession.Commit
```

The CreateBytesMessage Method

Create a BytesMessage.

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

The CreateMapMessage Method

Create a MapMessage.

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

The CreateMessage Method

Create a Message.

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

The CreateStreamMessage Method

Create a StreamMessage.

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

The CreateTextMessage Method

Create a TextMessage.

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

Name	Description
Text	The string used to initialize this message.

The Recover Method

Stops message delivery in this session, and restart sending messages with the oldest unacknowledged message.

```
XAQueueSession.Recover()
```

The Rollback Method

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

```
XAQueueSession.Rollback()
```

The Run Method

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

```
XAQueueSession.Run()
```

5.7.60. Properties of the QueueSender Object

The MessageListener Property

This property is not currently supported.

The QueueSession Property

Returns the queue session associated with this XAQueueSession.

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

The Transacted Property

Returns an indication that the session is in transacted mode.

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

5.7.61. 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).

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

```
XASession.Rollback
```

The Run Method

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

```
XASession.Run
```

5.7.62. Properties of the Session Object

The MessageListener Property

This property is currently not supported.

The Transacted Property

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

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

5.7.63. The XATopicConnection Object

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

The CreateTopicSession Method

Create a TopicSession

```
XATopicConnection.CreateTopicSession(Transacted As Boolean,  
acknowledgeMode As AcknowledgeModeConstants) As TopicSession
```

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

The Start Method

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

```
XATopicConnection.Start
```

The Stop Method

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

```
XATopicConnection.Stop
```

5.7.64. Properties of the TopicConnection

The ClientID Property

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

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

The MetaData Property

This property is currently not supported.

5.7.65. The XATopicConnectionFactory Object

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

The CreateTopicConnection Method

Create a topic connection with default user identity.

```
XATopicConnectionFactory.CreateTopicConnection() As TopicConnection
```

The CreateXATopicConnection Method

Create an XA topic connection with default user identity.

```
XATopicConnectionFactory.CreateTopicConnection() As TopicConnection
```

5.7.66. Properties of the TopicConnectionFactory

The HostName Property

The HostName property sets or returns the host name of the machine that the JMS IQ Server is running on.

```
XATopicConnectionFactory.HostName = String  
String = XATopicConnectionFactory.HostName
```

The Port Property

The Port property sets or returns the port number that the JMS IQ Server is listening on, default value is 7555

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

The PortOffset Property

The PortOffset sets or returns the port offset number of the JMS IQ Server if more than one Message Server is running on same host machine and using same port number.

```
XATopicConnectionFactory.PortOffset = Long  
Long = XATopicConnectionFactory
```

5.7.67. The XATopicSession Object

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

The Commit Method

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

```
XATopicSession.Commit
```

The CreateBytesMessage Method

The CreateBytesMessage method creates a BytesMessage.

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

The CreateMapMessage Method

The CreateMapMessage method creates a MapMessage.

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

The CreateMessage Method

Create a Message.

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

The CreateStreamMessage Method

Create a StreamMessage.

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

The CreateTextMessage Method

The CreateTextMessage method creates TextMessage.

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

Name	Description
text	The string used to initialize this message.

The Recover Method

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

```
XATopicSession.Recover
```

The Rollback Method

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

```
TopicSession.Rollback
```

The Run Method

The Run method is an optional method

```
TopicSession.Run
```

5.7.68. Properties of the TopicSession Object

The MessageListener Property

This property is currently not supported.

The TopicSession Property

Returns the topic session associated with this XATopicSession.

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

The Transacted Property

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

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

5.8 The C API for JMS

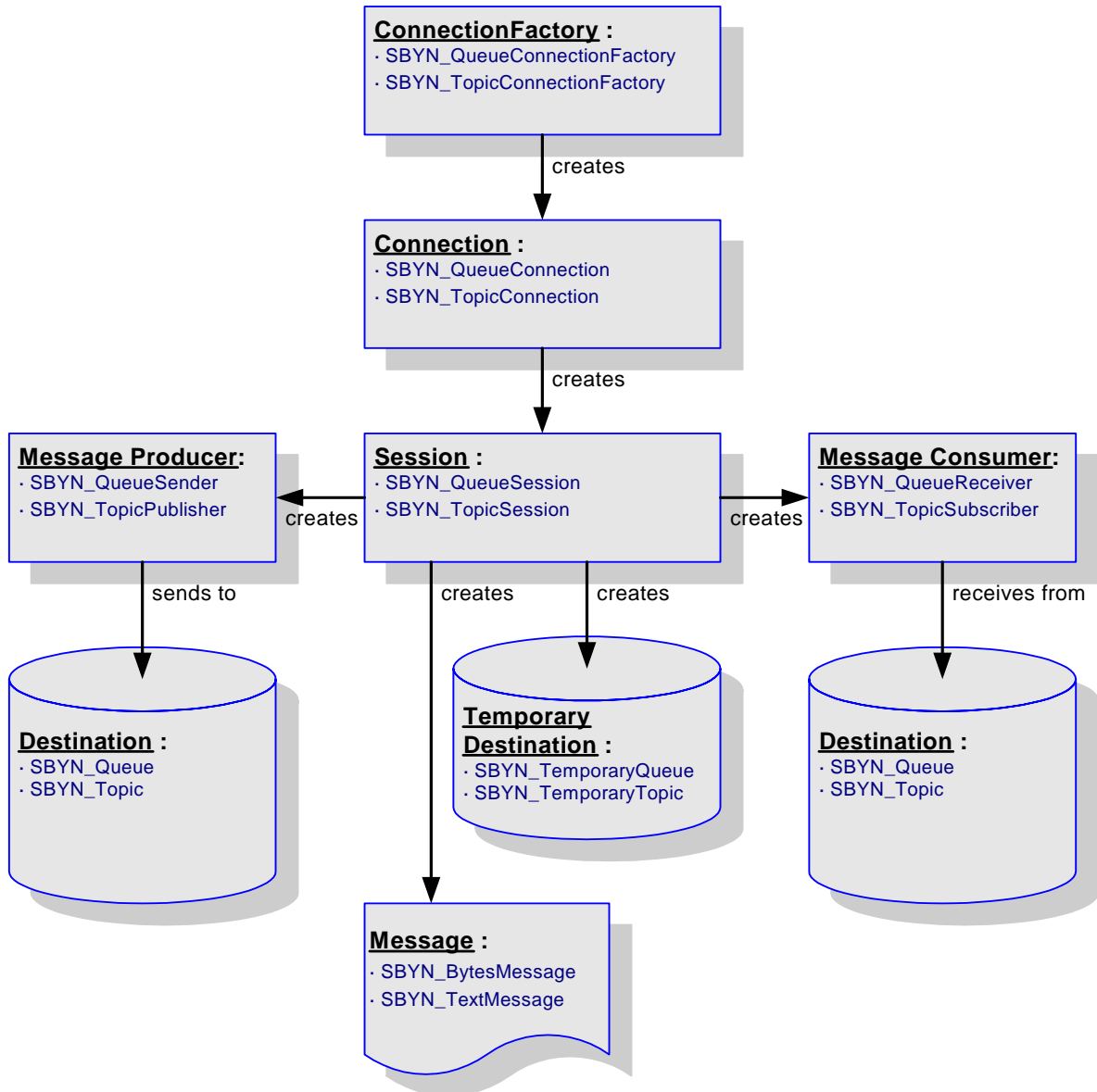
The C API for JMS, in the library `stc_msclient.dll`, is a wrapper around the C++ API for JMS. This section provides the following information:

- [Architectural Overview](#) on page 356
- [Structures](#) on page 357
- [Interfaces](#) on page 360
- Detailed information on the following:
 - ♦ Function prototypes for each interface
 - ♦ [Destructor Methods](#) on page 446
 - ♦ [The WString Helper Interface](#) on page 452
 - ♦ [The WStringList Helper Interface](#) on page 454
- [Error Codes and Messages in the C API for JMS](#) on page 455.

5.8.1. Architectural Overview

The standard JMS API object model is followed in C, as illustrated in Figure 34.

Figure 34 JMS C Object Model



5.8.2. Structures

The C API for JMS comprises the following structures:

Table 8 Structures in C API for JMS

For Point-to-Point (P2P) Messaging	For Publish/Subscribe (Pub/Sub) Messaging
struct SBYN_QueueConnectionFactory	struct SBYN_TopicConnectionFactory
struct SBYN_QueueConnection	struct SBYN_TopicConnection
struct SBYN_QueueSession	struct SBYN_TopicSession
struct SBYN_QueueSender	struct SBYN_TopicPublisher
struct SBYN_QueueReceiver	struct SBYN_TopicSubscriber
struct SBYN_Queue	struct SBYN_Topic
struct SBYN_TemporaryQueue	struct SBYN_TemporaryTopic
struct SBYN_QueueRequestor	struct SBYN_TopicRequestor
For P2P and Pub/Sub Messaging	
struct SBYN_Destination	
struct SBYN_Message	
struct SBYN_BytesMessage	
struct SBYN_TextMessage	
struct SBYN_ConnectionMetaData	
struct SBYN_WString	
struct SBYN_WStringList	

5.8.3. Constants

The C API for JMS defines values for the following types of constants:

- [DeliveryMode Constants](#) on page 358
- [DestinationType Constants](#) on page 358
- [MessageType Constants](#) on page 358
- [Session Constants](#) on page 358
- [Transacted Constants](#) on page 359
- [Miscellaneous Constants Setting Message Class Defaults](#) on page 360
- [Other Miscellaneous Constants](#) on page 360

DeliveryMode Constants

Table 9 Values for DeliveryMode Constants

Name	
SBYN_NON_PERSISTENT	0
SBYN_PERSISTENT	1

NON_PERSISTENT

0 signifies non-persistent delivery mode. This mode maximizes performance, and should be used if an occasional lost message is tolerable.

PERSISTENT

1 signifies persistent delivery mode. This mode maximizes reliability, and should be used if the application will have problems if the message is lost in transit.

DestinationType Constants

Table 10 Values for DestinationType Constants

Name	
SBYN_DESTINATION_TYPE_QUEUE	0
SBYN_DESTINATION_TYPE_TEMPORARYQUEUE	1
SBYN_DESTINATION_TYPE_TOPIC	2
SBYN_DESTINATION_TYPE_TEMPORARYTOPIC	3

MessageType Constants

Table 11 Values for MessageType Constants

Name	
SBYN_MESSAGE_TYPE_MESSAGE	0
SBYN_MESSAGE_TYPE_TEXT	1
SBYN_MESSAGE_TYPE_BYTES	2

Session Constants

Table 12 Values for Session Constants

Name	
SBYN_AUTO_ACKNOWLEDGE	1
SBYN_CLIENT_ACKNOWLEDGE	2

Table 12 Values for Session Constants

Name	
SBYN_DUPS_OK_ACKNOWLEDGE	3

AutoAcknowledge Mode

1 signifies auto-acknowledgment: The session automatically acknowledges a client's receipt of a message either upon its successful return from a call to **receive** or upon successful return of the `MessageListener` it has called to process the message.

ClientAcknowledge Mode

2 signifies acknowledgment by client: A client acknowledges a message by calling the message's **acknowledge** method. Acknowledging a consumed message automatically acknowledges the receipt of all messages that have been delivered by its session.

DupsOKAcknowledge Mode

3 indicates that duplicates are acceptable, and instructs the session to lazily acknowledge message delivery. This setting is likely to cause delivery of some duplicate messages if JMS fails, and should only be used by consumers that are tolerant of duplicate messages. The benefit is the reduction of session overhead, achieved by minimizing the work done to prevent duplicates.

Transacted Constants

Table 13 Values for Transacted Constants

Name	
SBYN_NON_TRANSACTED	0
SBYN_TRANSACTED	1

If a session is specified as being *transacted*, it supports a single series of transactions. A set of messages-received is grouped into an atomic unit of input, and a set of staged messages-to-be-sent is grouped into an atomic unit of output. When a transaction performs a *commit*, the atomic unit of input is acknowledged and the associated atomic unit of output is sent. If, instead, the transaction performs a *rollback*, all messages in the atomic unit of output are destroyed and the session's input is automatically recovered.

A transaction is completed only by a commit or by a rollback. The completion of a session's current transaction automatically begins the next transaction. In this way, a transacted session always has a current transaction within which work is done.

Miscellaneous Constants Setting Message Class Defaults

Table 14 Values for Miscellaneous Constants Setting Message Class Defaults

Name	
SBYN_DEFAULT_DELIVERY_MODE	1
SBYN_DEFAULT_PRIORITY	4
SBYN_DEFAULT_TIME_TO_LIVE	0

Default Setting for DeliveryMode

See [“DeliveryMode Constants” on page 358](#).

Default Setting for Priority

JMS defines a ten-level priority value: **0** is lowest priority (least expedited) and **9** is highest. Clients should consider priorities **0** through **4** as gradations of normal priority and priorities **5** through **9** as gradations of expedited priority.

Default Setting for TimeToLive

Length of time that a produced message should be retained by the message system. Measured in milliseconds elapsed since its dispatch time. The default, **0**, has the special meaning of “retain forever”—that is, the message never expires on its own.

Other Miscellaneous Constants

Table 15 Values for Other Miscellaneous Constants

Name	Value
DEFAULT_PORT	7555
DEFAULT_SERVER_NAME	“localhost”
MSCLIENT_DLL_NAME	“stc_msclient.dll”
FALSE	0
TRUE	1

5.8.4. Interfaces

The following interfaces have defined prototypes:

- [The Message Interface](#) on page 361
- [The Extended Message Interface](#) on page 381
- [BytesMessage Methods](#) on page 382
- [TextMessage Methods](#) on page 394
- [The QueueConnectionFactory Interface](#) on page 395

- [The Connection Interface](#) on page 396
- [The Session Interface](#) on page 400
- [The TopicConnectionFactory Interface](#) on page 412
- [The Destination Interface](#) on page 413
- [The QueueReceiver Interface](#) on page 415
- [The TopicSubscriber Interface](#) on page 418
- [The QueueSender Interface](#) on page 421
- [The TopicPublisher Interface](#) on page 432
- [The TopicRequestor Interface](#) on page 442
- [The QueueRequestor Interface](#) on page 444
- [Destructor Methods](#) on page 446
- [The WString Helper Interface](#) on page 452
- [The WStringList Helper Interface](#) on page 454

5.8.5. The Message Interface

The **Message** interface defines methods for working with a **Message** object. The interface includes the following methods:

- [Acknowledge](#) on page 362
- [ClearBody](#) on page 363
- [ClearProperties](#) on page 363
- [PropertyExists](#) on page 364
- [GetBooleanProperty](#) on page 364
- [GetByteProperty](#) on page 365
- [GetDoubleProperty](#) on page 365
- [GetFloatProperty](#) on page 366
- [GetIntProperty](#) on page 366
- [GetLongProperty](#) on page 367
- [GetShortProperty](#) on page 367
- [GetStringProperty](#) on page 368
- [SetBooleanProperty](#) on page 368
- [SetByteProperty](#) on page 369
- [SetDoubleProperty](#) on page 369
- [SetFloatProperty](#) on page 370
- [SetIntProperty](#) on page 370

- [SetLongProperty](#) on page 371
- [SetShortProperty](#) on page 371
- [SetStringProperty](#) on page 372
- [GetJMSCorrelationID](#) on page 372
- [GetJMSCorrelationIDAsBytes](#) on page 372
- [GetJMSDeliveryMode](#) on page 373
- [GetJMSExpiration](#) on page 373
- [GetJMSMessageID](#) on page 374
- [GetJMSPriority](#) on page 374
- [GetJMSRedelivered](#) on page 375
- [GetJMSReplyTo](#) on page 375
- [GetJMSTimestamp](#) on page 376
- [GetJMSType](#) on page 376
- [SetJMSCorrelationID](#) on page 377
- [SetJMSCorrelationIDAsBytes](#) on page 377
- [SetJMSDeliveryMode](#) on page 378
- [SetJMSExpiration](#) on page 378
- [SetJMSMessageID](#) on page 378
- [SetJMSPriority](#) on page 379
- [SetJMSRedelivered](#) on page 379
- [SetJMSReplyTo](#) on page 380
- [SetJMSTimestamp](#) on page 380
- [SetJMSType](#) on page 381
- [GetMessageType](#) on page 381

Acknowledge

Syntax

`Acknowledge(pMsg, iError, pczError)`

Description

Acknowledges the receipt of current and previous messages.

Parameters

Name	Type	Description
pMsg	SBYN_Message*	Pointer to the message.

Name	Type	Description
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

None.

ClearBody

Syntax

```
ClearBody(pMsg, iError, pczError)
```

Description

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

Parameters

Name	Type	Description
pMsg	SBYN_Message*	Pointer to the message.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

None.

ClearProperties

Syntax

```
ClearProperties(pMsg, iError, pczError)
```

Description

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

Parameters

Name	Type	Description
pMsg	SBYN_Message*	Pointer to the message.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

None.

PropertyExists

Syntax

```
PropertyExists(pMsg, pczName, iError, pczError)
```

Description

Checks whether a value for a specific property exists.

Parameters

Name	Type	Description
pMsg	SBYN_Message*	Pointer to the message.
pczName	char*	Pointer to the name of the property to check.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

bool

Returns **true** if the property value is defined; otherwise, returns **false**.

GetBooleanProperty

Syntax

```
GetBooleanProperty(pMsg, pczName, iError, pczError)
```

Description

Reads the value of the specified Boolean property in the specified message.

Parameters

Name	Type	Description
pMsg	SBYN_Message*	Pointer to the message.
pczName	char*	Pointer to the name of the property to check.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

SBYN_BOOL

The value of the property.

GetByteProperty

Syntax

```
GetByteProperty(pMsg, pczName, iError, pczError)
```

Description

Reads the value of the specified byte property in the specified message.

Parameters

Name	Type	Description
pMsg	SBYN_Message*	Pointer to the message.
pczName	char*	Pointer to the name of the property to check.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

unsigned char

The value of the property.

GetDoubleProperty

Syntax

```
GetDoubleProperty(pMsg, pczName, iError, pczError)
```

Description

Reads the value of the specified double (numeric) property in the specified message.

Parameters

Name	Type	Description
pMsg	SBYN_Message*	Pointer to the message.
pczName	char*	Pointer to the name of the property to check.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

double

The value of the property.

GetFloatProperty

Syntax

```
GetFloatProperty(pMsg, pczName, iError, pczError)
```

Description

Reads the value of the specified floating-point (numeric) property in the specified message.

Parameters

Name	Type	Description
pMsg	SBYN_Message*	Pointer to the message.
pczName	char*	Pointer to the name of the property to check.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

float

The value of the property.

GetIntProperty

Syntax

```
GetIntProperty(pMsg, pczName, iError, pczError)
```

Description

Reads the value of the specified integer (numeric) property in the specified message.

Parameters

Name	Type	Description
pMsg	SBYN_Message*	Pointer to the message.
pczName	char*	Pointer to the name of the property to check.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

int

The value of the property.

GetLongProperty

Syntax

```
GetLongProperty(pMsg, pczName, iError, pczError)
```

Description

Reads the value of the specified long (numeric) property in the specified message.

Parameters

Name	Type	Description
pMsg	SBYN_Message*	Pointer to the message.
pczName	char*	Pointer to the name of the property to check.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

long
The value of the property.

GetShortProperty

Syntax

```
GetShortProperty(pMsg, pczName, iError, pczError)
```

Description

Reads the value of the specified short (numeric) property in the specified message.

Parameters

Name	Type	Description
pMsg	SBYN_Message*	Pointer to the message.
pczName	char*	Pointer to the name of the property to check.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

short
The value of the property.

GetStringProperty

Syntax

```
GetStringProperty(pMsg, pczName, iError, pczError)
```

Description

Reads the value of the specified text property in the specified message.

Parameters

Name	Type	Description
pMsg	SBYN_Message*	Pointer to the message.
pczName	char*	Pointer to the name of the property to check.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

WString*

Pointer to a **WString** (wide string) object containing the value of the property.

SetBooleanProperty

Syntax

```
SetBooleanProperty(pMsg, pczName, bValue, iError, pczError)
```

Description

Writes a value for the specified Boolean property in the specified message.

Parameters

Name	Type	Description
pMsg	SBYN_Message*	Pointer to the message.
pczName	char*	Pointer to the property name.
bValue	bool	The value to be written.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

None.

SetByteProperty

Syntax

```
SetByteProperty(pMsg, pczName, cValue, iError, pczError)
```

Description

Writes a value for the specified byte property in the specified message.

Parameters

Name	Type	Description
pMsg	SBYN_Message*	Pointer to the message.
pczName	char*	Pointer to the property name.
cValue	unsigned char	The value to be written.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

None.

SetDoubleProperty

Syntax

```
SetDoubleProperty(pMsg, pczName, dblValue, iError, pczError)
```

Description

Writes a value for the specified double (numeric) property in the specified message.

Parameters

Name	Type	Description
pMsg	SBYN_Message*	Pointer to the message.
pczName	char*	Pointer to the property name.
dblValue	double	The value to be written.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

None.

SetFloatProperty

Syntax

```
SetFloatProperty(pMsg, pczName, fltValue, iError, pczError)
```

Description

Writes a value for the specified floating-point (numeric) property in the specified message.

Parameters

Name	Type	Description
pMsg	SBYN_Message*	Pointer to the message.
pczName	char*	Pointer to the property name.
fltValue	float	The value to be written.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

None.

SetIntProperty

Syntax

```
SetIntProperty(pMsg, pczName, iValue, iError, pczError)
```

Description

Writes a value for the specified integer (numeric) property in the specified message.

Parameters

Name	Type	Description
pMsg	SBYN_Message*	Pointer to the message.
pczName	char*	Pointer to the property name.
iValue	int	The value to be written.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

None.

SetLongProperty

Syntax

```
SetLongProperty(pMsg, pczName, lValue, iError, pczError)
```

Description

Writes a value for the specified long (numeric) property in the specified message.

Parameters

Name	Type	Description
pMsg	SBYN_Message*	Pointer to the message.
pczName	char*	Pointer to the property name.
lValue	long	The value to be written.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

None.

SetShortProperty

Syntax

```
SetShortProperty(pMsg, pczName, nValue, iError, pczError)
```

Description

Writes a value for the specified short (numeric) property in the specified message.

Parameters

Name	Type	Description
pMsg	SBYN_Message*	Pointer to the message.
pczName	char*	Pointer to the property name.
nValue	short	The value to be written.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

None.

SetStringProperty

Syntax

```
SetStringProperty(pMsg, pczName, pczValue, iError, pczError)
```

Description

Writes a value for the specified text property in the specified message.

Parameters

Name	Type	Description
pMsg	SBYN_Message*	Pointer to the message.
pczName	char*	Pointer to the property name.
pczValue	char*	The value to be written.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

None.

GetJMSCorrelationID

Syntax

```
GetJMSCorrelationID(pMsg, iError, pczError)
```

Description

Retrieves the correlation ID for the specified message.

Parameters

Name	Type	Description
pMsg	SBYN_Message*	Pointer to the message.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

WString*

Pointer to a **WString** (wide string) object containing the text.

GetJMSCorrelationIDAsBytes

Syntax

```
GetJMSCorrelationIDAsBytes(pMsg, iError, pczError)
```

Description

Retrieves the correlation ID for the specified message as an array of characters.

Parameters

Name	Type	Description
pMsg	SBYN_Message*	Pointer to the message.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

char*

Pointer to an array of characters containing the text.

GetJMSDeliveryMode

Syntax

```
GetJMSDeliveryMode(pMsg, iError, pczError)
```

Description

Retrieves the value of the **DeliveryMode** property for the specified message.

Parameters

Name	Type	Description
pMsg	SBYN_Message*	Pointer to the message.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

int

See [“DeliveryMode Constants” on page 358](#).

GetJMSExpiration

Syntax

```
GetJMSExpiration(pMsg, iError, pczError)
```

Description

Retrieves the value of the timestamp set for the expiration of the specified message.

Parameters

Name	Type	Description
pMsg	SBYN_Message*	Pointer to the message.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

long
Timestamp.

GetJMSMessageID

Syntax

```
GetJMSMessageID(pMsg, iError, pczError)
```

Description

Retrieves the message ID of the specified message.

Parameters

Name	Type	Description
pMsg	SBYN_Message*	Pointer to the message.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

WString*
Pointer to a **WString** (wide string) object containing the text.

GetJMSPriority

Syntax

```
GetJMSPriority(pMsg, iError, pczError)
```

Description

Retrieves the priority level for the specified message.

Parameters

Name	Type	Description
pMsg	SBYN_Message*	Pointer to the message.
iError	int*	Pointer to the error number.

Name	Type	Description
pczError	char*	Pointer to the error message text.

Return Value

int

The priority level.

Also see [“Miscellaneous Constants Setting Message Class Defaults” on page 360.](#)

GetJMSRedelivered

Syntax

```
GetJMSRedelivered(pMsg, iError, pczError)
```

Description

Retrieves an indication of whether the specified message is being redelivered.

Parameters

Name	Type	Description
pMsg	SBYN_Message*	Pointer to the message.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

SBYN_BOOL

Returns **true** if the message is being redelivered; otherwise, returns **false**.

GetJMSReplyTo

Syntax

```
GetJMSReplyTo(pMsg, iError, pczError)
```

Description

Retrieves the **Destination** object where a reply to the specified message should be sent (for request/reply messaging).

Parameters

Name	Type	Description
pMsg	SBYN_Message*	Pointer to the message.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

SBYN_Destination*

Pointer to the **Destination** object.

GetJMSTimestamp

Syntax

```
GetJMSTimestamp(pMsg, iError, pczError)
```

Description

Retrieves the timestamp of the specified message.

Parameters

Name	Type	Description
pMsg	SBYN_Message*	Pointer to the message.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

long

Timestamp.

GetJMSType

Syntax

```
GetJMSType(pMsg, iError, pczError)
```

Description

Gets the message type identifier supplied by the client when the message was sent.

Parameters

Name	Type	Description
pMsg	SBYN_Message*	Pointer to the message.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

WString*

Pointer to a **WString** (wide string) object containing the text.

SetJMSCorrelationID

Syntax

```
SetJMSCorrelationID(pMsg, pczValue, iError, pczError)
```

Description

Sets the correlation ID for the specified message.

Parameters

Name	Type	Description
pMsg	SBYN_Message*	Pointer to the message.
pczValue	char*	Pointer to the text string containing the correlation ID.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

None.

SetJMSCorrelationIDAsBytes

Syntax

```
SetJMSCorrelationIDAsBytes(pMsg, pczValue, iError, pczError)
```

Description

Sets the correlation ID for the specified message.

Parameters

Name	Type	Description
pMsg	SBYN_Message*	Pointer to the message.
pczValue	char*	Pointer to the character array containing the bytes for the correlation ID.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

None.

SetJMSDeliveryMode

Syntax

```
SetJMSDeliveryMode(pMsg, value, iValue, iError, pczError)
```

Description

Sets the delivery mode for the specified message. See [“DeliveryMode Constants” on page 358](#).

Parameters

Name	Type	Description
pMsg	SBYN_Message*	Pointer to the message.
iValue	int	Value corresponding to the delivery mode.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

None.

SetJMSExpiration

Syntax

```
SetJMSExpiration(pMsg, lValue, iError, pczError)
```

Description

Sets the timestamp at which the specified message is due to expire.

Parameters

Name	Type	Description
pMsg	SBYN_Message*	Pointer to the message.
lValue	long	Timestamp of the expiration.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

None.

SetJMSMessageID

Syntax

```
SetJMSMessageID(pMsg, pczValue, iError, pczError)
```

Description

Sets the message ID of the specified message.

Parameters

Name	Type	Description
pMsg	SBYN_Message*	Pointer to the message.
pczValue	char*	Pointer to the text string containing the message ID.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

None.

SetJMSPriority

Syntax

```
SetJMSPriority(pMsg, iValue, iError, pczError)
```

Description

Sets the priority level for the specified message.

Parameters

Name	Type	Description
pMsg	SBYN_Message*	Pointer to the message.
iValue	int	The priority level.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

None.

SetJMSRedelivered

Syntax

```
SetJMSRedelivered(pMsg, fValue, iError, pczError)
```

Description

Determines whether to flag the specified message as being redelivered. Used, for example, to specify redelivery for a message that has been sent but not acknowledged.

Parameters

Name	Type	Description
pMsg	SBYN_Message*	Pointer to the message.
fValue	SBYN_BOOL	Flag: If true , the message is being redelivered.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

None.

SetJMSReplyTo

Syntax

```
SetJMSReplyTo(pMsg, pDest, iError, pczError)
```

Description

Sets the **Destination** object where a reply to the specified message should be sent.

Parameters

Name	Type	Description
pMsg	SBYN_Message*	Pointer to the message.
pDest	SBYN_Destination*	Pointer to the Destination object.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

None.

SetJMSTimestamp

Syntax

```
SetJMSTimestamp(pMsg, lValue, iError, pczError)
```

Description

Sets the timestamp (JMSTimestamp header field) that the specified message was handed off to a provider to be sent. Note that this is not necessarily the time the message is actually transmitted; the actual **send** can occur later because of transactions or other client-side queuing of messages.

Parameters

Name	Type	Description
pMsg	SBYN_Message*	Pointer to the message.
lValue	long	Timestamp to be set.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

None.

SetJMSType

Syntax

```
SetJMSType(pMsg, pczJMSType, iError, pczError)
```

Description

Sets the JMSType header field, which is often used to identify the message structure and the payload type.

Parameters

Name	Type	Description
pMsg	SBYN_Message*	Pointer to the message.
pczJMSType	char*	Pointer to the text string containing the data.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

None.

5.8.6. The Extended Message Interface

The extended message interface includes the following function:

- [GetMessageType](#) on page 381

GetMessageType

Syntax

```
GetMessageType(pMsg)
```

Description

Retrieves the message type (bytesmessage, textmessage, and so forth).

Parameters

Name	Type	Description
pMsg	SBYN_Message*	Pointer to the message.

Return Value

MessageType_t

See ["MessageType Constants" on page 358](#).

5.8.7. BytesMessage Methods

A **BytesMessage** object is used for messages containing a stream of uninterpreted bytes. The receiver of the message supplies the interpretation of the bytes.

If you are familiar with the Java implementation, note that methods **ReadBytes** and **WriteBytes** differ from the standard JMS specification; see [Differences Between Java and C in the BytesMessage Interface](#) on page 456.

The **BytesMessage** methods include the following:

- [ReadBoolean](#) on page 383
- [ReadByte](#) on page 383
- [ReadBytes](#) on page 384
- [ReadChar](#) on page 384
- [ReadDouble](#) on page 385
- [ReadFloat](#) on page 385
- [ReadInt](#) on page 386
- [ReadLong](#) on page 386
- [ReadShort](#) on page 386
- [ReadUnsignedByte](#) on page 387
- [ReadUnsignedShort](#) on page 387
- [ReadUTF](#) on page 388
- [Reset](#) on page 388
- [WriteBoolean](#) on page 389
- [WriteByte](#) on page 389
- [WriteBytes](#) on page 390
- [WriteBytesEx](#) on page 390
- [WriteChar](#) on page 391
- [WriteDouble](#) on page 391
- [WriteFloat](#) on page 392

- [WriteInt](#) on page 392
- [WriteLong](#) on page 392
- [WriteShort](#) on page 393
- [WriteUTF](#) on page 393

ReadBoolean

Syntax

```
ReadBoolean(pMsg, iError, pczError)
```

Description

Reads a Boolean value from the **BytesMessage** stream.

Parameters

Name	Type	Description
pMsg	SBYN_Message*	Pointer to the message.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

SBYN_BOOL

The value read from the **BytesMessage** stream.

ReadByte

Syntax

```
ReadByte(pMsg, iError, pczError)
```

Description

Reads a single unsigned character from the **BytesMessage** stream.

Parameters

Name	Type	Description
pMsg	SBYN_Message*	Pointer to the message.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

unsigned char

The value read from the **BytesMessage** stream.

ReadBytes

Syntax

```
ReadBytes(pMsg, pczValue, iLength, iError, pczError)
```

Description

Reads a portion of the **BytesMessage** stream into a buffer. If the length of array value is less than the bytes remaining to be read from the stream, the array is filled.

A subsequent call reads the next increment, and so on.

If there are fewer bytes remaining in the stream than the length of array value, the bytes are read into the array, and the return value (total number of bytes read) is less than the length of the array, indicating that there are no more bytes left to be read from the stream. The next read of the stream returns **-1**.

Parameters

Name	Type	Description
pMsg	SBYN_Message*	Pointer to the message.
pczValue	unsigned char*	Pointer to the buffer into which the bytes are read.
iLength	int	The number of bytes to read; must be less than the length of the buffer.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

int

Number of bytes read into the buffer; or **-1** if all bytes were read previously.

ReadChar

Syntax

```
ReadChar(pMsg, iError, pczError)
```

Description

Reads a single Unicode character from the **BytesMessage** stream.

Parameters

Name	Type	Description
pMsg	SBYN_Message*	Pointer to the message.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

unsigned short

The value read from the **BytesMessage** stream.

ReadDouble

Syntax

```
ReadDouble(pMsg, iError, pczError)
```

Description

Reads a double numeric value from the **BytesMessage** stream.

Parameters

Name	Type	Description
pMsg	SBYN_Message*	Pointer to the message.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

double

The value read from the **BytesMessage** stream.

ReadFloat

Syntax

```
ReadFloat(pMsg, iError, pczError)
```

Description

Reads a floating-point numeric value from the **BytesMessage** stream.

Parameters

Name	Type	Description
pMsg	SBYN_Message*	Pointer to the message.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

float

The value read from the **BytesMessage** stream.

ReadInt

Syntax

```
ReadInt(pMsg, iError, pczError)
```

Description

Reads a signed integer value from the **BytesMessage** stream.

Parameters

Name	Type	Description
pMsg	SBYN_Message*	Pointer to the message.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

int

The value read from the **BytesMessage** stream.

ReadLong

Syntax

```
ReadLong(pMsg, iError, pczError)
```

Description

Reads a signed long integer from the **BytesMessage** stream.

Parameters

Name	Type	Description
pMsg	SBYN_Message*	Pointer to the message.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

long

The value read from the **BytesMessage** stream.

ReadShort

Syntax

```
ReadShort(pMsg, iError, pczError)
```

Description

Reads a signed short integer from the **BytesMessage** stream.

Parameters

Name	Type	Description
pMsg	SBYN_Message*	Pointer to the message.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

short

The value read from the **BytesMessage** stream.

ReadUnsignedByte

Syntax

```
ReadUnsignedByte(pMsg, iError, pczError)
```

Description

Reads an unsigned short integer from the **BytesMessage** stream.

Parameters

Name	Type	Description
pMsg	SBYN_Message*	Pointer to the message.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

int

The value read from the **BytesMessage** stream.

ReadUnsignedShort

Syntax

```
ReadUnsignedShort(pMsg, iError, pczError)
```

Description

Reads an unsigned short integer from the **BytesMessage** stream.

Parameters

Name	Type	Description
pMsg	SBYN_Message*	Pointer to the message.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

int

The value read from the **BytesMessage** stream.

ReadUTF

Syntax

```
ReadUTF(pMsg, iError, pczError)
```

Description

Reads the value of a string that has been encoded using a modified UTF-8 format from the **BytesMessage** stream.

Parameters

Name	Type	Description
pMsg	SBYN_Message*	Pointer to the message.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

WString*

Pointer to a **WString** (wide string) object containing the text from the **BytesMessage** stream.

Reset

Syntax

```
Reset(pMsg, iError, pczError)
```

Description

Puts the message body into read-only mode and repositions the stream of bytes to the beginning.

Parameters

Name	Type	Description
pMsg	SBYN_Message*	Pointer to the message.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

None.

WriteBoolean

Syntax

```
WriteBoolean(pMsg, fValue, iError, pczError)
```

Description

Writes a Boolean value to the **BytesMessage** stream.

Parameters

Name	Type	Description
pMsg	SBYN_Message*	Pointer to the message.
fValue	SBYN_BOOL	The value to be written.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

None.

WriteByte

Syntax

```
WriteByte(pMsg, cValue, iError, pczError)
```

Description

Writes a single byte (unsigned char) to the **BytesMessage** stream.

Parameters

Name	Type	Description
pMsg	SBYN_Message*	Pointer to the message.
cValue	unsigned char	The value to be written.
iError	int*	Pointer to the error number.

Name	Type	Description
pczError	char*	Pointer to the error message text.

Return Value

None.

WriteBytes

Syntax

```
WriteBytes(pMsg, pczValue, iError, pczError)
```

Description

Writes an array of bytes (unsigned char values) to the **BytesMessage** stream.

Parameters

Name	Type	Description
pMsg	SBYN_Message*	Pointer to the message.
pczValue	unsigned char*	Pointer to the value to be written.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

None.

WriteBytesEx

Syntax

```
WriteBytesEx(pMsg, pczValue, iOffset, iLength, iError, pczError)
```

Description

Writes a portion of a byte array (unsigned char values) to the **BytesMessage** stream. For example, to extract "nag" from "manager", set iOffset=2 and iLength=3.

Parameters

Name	Type	Description
pMsg	SBYN_Message*	Pointer to the message.
pczValue	unsigned char*	Pointer to the value to be written.
iOffset	int	The initial offset within the byte array.
iLength	int	The number of bytes to use.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

None.

WriteChar

Syntax

```
WriteChar(pMsg, cValue, iError, pczError)
```

Description

Writes an unsigned short integer to the **BytesMessage** stream.

Parameters

Name	Type	Description
pMsg	SBYN_Message*	Pointer to the message.
cValue	unsigned short	The value to be written.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

None.

WriteDouble

Syntax

```
WriteDouble(pMsg, dblValue, iError, pczError)
```

Description

Writes a double numeric value to the **BytesMessage** stream.

Parameters

Name	Type	Description
pMsg	SBYN_Message*	Pointer to the message.
dblValue	double	The value to be written.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

None.

WriteFloat

Syntax

```
WriteFloat(pMsg, floatValue, iError, pczError)
```

Description

Writes a floating-point numeric value to the **BytesMessage** stream.

Parameters

Name	Type	Description
pMsg	SBYN_Message*	Pointer to the message.
floatValue	float	The value to be written.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

None.

WriteInt

Syntax

```
WriteInt(pMsg, iValue, iError, pczError)
```

Description

Writes an integer numeric value to the **BytesMessage** stream.

Parameters

Name	Type	Description
pMsg	SBYN_Message*	Pointer to the message.
iValue	int	The value to be written.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

None.

WriteLong

Syntax

```
WriteLong(pMsg, lValue, iError, pczError)
```


Description

Writes a long numeric value to the **BytesMessage** stream.

Parameters

Name	Type	Description
pMsg	SBYN_Message*	Pointer to the message.
lValue	int	The value to be written.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

None.

WriteShort

Syntax

```
WriteShort(pMsg, nValue, iError, pczError)
```

Description

Writes a short numeric value to the **BytesMessage** stream.

Parameters

Name	Type	Description
pMsg	SBYN_Message*	Pointer to the message.
nValue	short	The value to be written.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

None.

WriteUTF

Syntax

```
WriteUTF(pMsg, pczValue, iError, pczError)
```

Description

Writes a character string to the **BytesMessage** stream using UTF-8 encoding.

Parameters

Name	Type	Description
pMsg	SBYN_Message*	Pointer to the message.
pczValue	char*	Pointer to the value to be written.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

None.

5.8.8. TextMessage Methods

A TextMessage is used to send a message containing text. It adds a text message body. When a client receives a TextMessage, it is in read-only mode. If an attempt is made to write to the message while in read-only mode, an error is returned. However, if ClearBody is called first, then message can be then read from and written to.

The TextMessage functions include the following:

- [GetText](#) on page 394
- [SetText](#) on page 395

GetText

Syntax

```
GetText(pMsg, iError, pczError)
```

Description

Retrieves the string containing the data associated with the message.

Parameters

Name	Type	Description
pMsg	SBYN_Message*	Pointer to the message.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

char*

Pointer to the string. The default value is 0 (null).

SetText

Syntax

```
SetText(pMsg, pczBuffer, iError, pczError)
```

Description

Sets the string containing the data associated with the message.

Parameters

Name	Type	Description
pMsg	SBYN_Message*	Pointer to the message.
pczBuffer	char*	Pointer to the text string.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

None.

5.8.9. The QueueConnectionFactory Interface

Using point-to-point messaging, a client uses a **QueueConnectionFactory** object to create **QueueConnection** objects.

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

- [CreateQueueConnectionFactory](#) on page 395
- [CreateQueueConnection](#) on page 396

CreateQueueConnectionFactory

Syntax

```
CreateQueueConnectionFactory(pczHost, iPort, iError, pczError)
```

Description

Constructs a **QueueConnectionFactory** object for the specified host and port. Once constructed, it can create **QueueConnection** objects for a point-to-point JMS provider.

Parameters

Name	Type	Description
pczHost	char*	Pointer to the text of the host name.
iPort	int	Port number.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

SBYN_QueueConnectionFactory*

Pointer to the **QueueConnectionFactory** object that was created.

CreateQueueConnection

Syntax

```
CreateQueueConnection(pQueCxnFac, iError, pczError)
```

Description

Constructs a **Connection** object; see [“The Connection Interface” on page 396](#).

Parameters

Name	Type	Description
pQueCxnFac	SBYN_QueueConnectionFactory*	Pointer to the QueueConnectionFactory object creating the QueueConnection .
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

SBYN_Connection* pointer.

5.8.10. The Connection Interface

A **Connection** object is an active connection to a JMS point-to-point provider or an active connection to a JMS pub/sub provider. A client uses a **Connection** to create one or more **Sessions** for producing and consuming messages.

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

- [ConnectionClose](#) on page 397
- [ConnectionGetClientID](#) on page 397
- [ConnectionSetClientID](#) on page 397
- [ConnectionSetClientID](#) on page 397
- [ConnectionStart](#) on page 398
- [ConnectionStop](#) on page 398
- [ConnectionCreateQueueSession](#) on page 399
- [ConnectionCreateTopicSession](#) on page 399

ConnectionClose

Syntax

```
ConnectionClose(pConn, iError, pczError)
```

Description

Closes the specified connection.

Parameters

Name	Type	Description
<i>pConn</i>	SBYN_Connection*	Pointer to the Connection object.
<i>iError</i>	int*	Pointer to the error number.
<i>pczError</i>	char*	Pointer to the error message text.

Return Value

None.

ConnectionGetClientID

Syntax

```
ConnectionGetClientID(pConn, iError, pczError)
```

Description

Retrieves the client ID associated with the specified **Connection** object.

Parameters

Name	Type	Description
<i>pConn</i>	SBYN_Connection*	Pointer to the Connection object.
<i>iError</i>	int*	Pointer to the error number.
<i>pczError</i>	char*	Pointer to the error message text.

Return Value

WString*

Pointer to a **WString** (wide string) object containing the text.

ConnectionSetClientID

Syntax

```
ConnectionSetClientID(pConn, pczClientID, iError, pczError)
```

Description

Sets the client ID to be associated with the specified **Connection** object. In other words, this method allows you to name the connection.

Parameters

Name	Type	Description
pConn	SBYN_Connection*	Pointer to the Connection object.
pczClientID	char*	Pointer to the text string for the client ID.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

None.

ConnectionStart

Syntax

```
ConnectionStart(pConn, iError, pczError)
```

Description

Starts (or restarts) delivering incoming messages via the specified **Connection** object. If the connection is already started, the call is ignored without error.

Parameters

Name	Type	Description
pConn	SBYN_Connection*	Pointer to the Connection object.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

None.

ConnectionStop

Syntax

```
ConnectionStop(pConn, iError, pczError)
```

Description

Temporarily halts delivering incoming messages via the specified **Connection** object. If the connection is already stopped, the call is ignored without error.

Parameters

Name	Type	Description
pConn	SBYN_Connection*	Pointer to the Connection object.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

None.

ConnectionCreateQueueSession

Syntax

```
ConnectionCreateQueueSession(pConn, fTransacted, iAckMode,
                             iError, pczError)
```

Description

Creates a **Session** object; see [“The Session Interface” on page 400](#).

Parameters

Name	Type	Description
pConn	SBYN_Connection*	Pointer to the Connection object.
fTransacted	SBYN_BOOL	Flag: If true , the session is transacted.
iAckMode	int	Ignored if the session is transacted. If the session is not transacted, indicates whether the consumer or client acknowledges messages that it receives; see Session Constants on page 358.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

SBYN_Session* pointer.

ConnectionCreateTopicSession

Syntax

```
ConnectionCreateTopicSession(pCxn, fTransacted, iAckMode,
                              iError, pczError)
```

Description

Creates a **TopicSession** object.

Parameters

Name	Type	Description
pCxn	SBYN_Connection*	Pointer to the Connection object.
fTransacted	SBYN_BOOL	Flag: If true , the session is transacted
iAckMode	int	Ignored if the session is transacted. If the session is not transacted, indicates whether the consumer or client acknowledges messages that it receives; see Session Constants on page 358.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

SBYN_Session* pointer.

5.8.11. The Session Interface

The Session interface includes the following methods:

- [SessionClose](#) on page 401
- [SessionCommit](#) on page 401
- [SessionGetTransacted](#) on page 402
- [SessionRecover](#) on page 402
- [SessionRollback](#) on page 402
- [SessionCreateBytesMessage](#) on page 403
- [SessionCreateTextMessage](#) on page 403
- [SessionCreateTextMessageEx](#) on page 404
- [SessionCreateQueue](#) on page 404
- [SessionCreateReceiver](#) on page 405
- [SessionCreateReceiveMessageSelector](#) on page 405
- [SessionCreateSender](#) on page 406
- [SessionCreateTemporaryQueue](#) on page 406
- [SessionCreateDurableSubscriber](#) on page 407
- [SessionCreateDurableSubscriberMessageSelector](#) on page 408
- [SessionCreatePublisher](#) on page 408
- [SessionCreateSubscriber](#) on page 409
- [SessionCreateSubscriberMessageSelector](#) on page 409
- [SessionCreateTemporaryTopic](#) on page 410

- [SessionCreateTopic](#) on page 411
- [SessionUnsubscribe](#) on page 411

SessionClose

Syntax

```
SessionClose(pSessn, iError, pczError)
```

Description

Closes the specified session.

Note: Sessions should be closed when they are no longer needed.

Parameters

Name	Type	Description
pSessn	SBYN_Session*	Pointer to the Session object.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

None.

SessionCommit

Syntax

```
SessionCommit(pSessn, iError, pczError)
```

Description

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

Parameters

Name	Type	Description
pSessn	SBYN_Session*	Pointer to the Session object.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

None.

SessionGetTransacted

Syntax

```
SessionGetTransacted(pSessn, iError, pczError)
```

Description

Queries whether the specified session is or is not transacted.

Parameters

Name	Type	Description
<i>pSessn</i>	SBYN_Session*	Pointer to the Session object.
<i>iError</i>	int*	Pointer to the error number.
<i>pczError</i>	char*	Pointer to the error message text.

Return Value

SBYN_BOOL

Returns **true** if the session is transacted; otherwise, returns **false**.

SessionRecover

Syntax

```
SessionRecover(pSessn, iError, pczError)
```

Description

Stops message delivery in the specified session, causes all messages that might have been delivered but not acknowledged to be marked as **redelivered**, and restarts message delivery with the oldest unacknowledged message. Note that redelivered messages need not be delivered in the exact order they were originally delivered.

Parameters

Name	Type	Description
<i>pSessn</i>	SBYN_Session*	Pointer to the Session object.
<i>iError</i>	int*	Pointer to the error number.
<i>pczError</i>	char*	Pointer to the error message text.

Return Value

None.

SessionRollback

Syntax

```
SessionRollback(pSessn, iError, pczError)
```

Description

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

Parameters

Name	Type	Description
pSessn	SBYN_Session*	Pointer to the Session object.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

None.

SessionCreateBytesMessage

Syntax

```
SessionCreateBytesMessage(pSessn, iError, pczError)
```

Description

Creates a **BytesMessage**— an object used to send a message containing a stream of uninterpreted bytes.

Parameters

Name	Type	Description
pSessn	SBYN_Session*	Pointer to the Session object.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

SBYN_Message*
Pointer to the created message object.

SessionCreateTextMessage

Syntax

```
SessionCreateTextMessage(pSessn, iError, pczError)
```

Description

Creates an uninitialized **TextMessage**— an object used to send a message containing a string to be supplied. Also see [“SessionCreateTextMessageEx” on page 404](#).

Parameters

Name	Type	Description
pSessn	SBYN_Session*	Pointer to the Session object.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

SBYN_Message*
Pointer to the created message object.

SessionCreateTextMessageEx

Syntax

```
SessionCreateTextMessageEx(pSessn, pczText, iError, pczError)
```

Description

Creates an initialized **TextMessage**— an object used to send a message containing the supplied string. Also see [“SessionCreateTextMessage” on page 403](#).

Parameters

Name	Type	Description
pQueSessn	SBYN_QueueSession*	Pointer to the QueueSession object.
pczText	char*	Pointer to the text string with which to initialize the message.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

SBYN_Message*
Pointer to the created message object.

SessionCreateQueue

Syntax

```
SessionCreateQueue(pSessn, pczQueName, iError, pczError)
```

Description

Creates an identity with a specific queue name; does not create a physical queue.

This functionality is provided for rare cases where clients need to dynamically create a queue identity with a provider-specific name. Clients that depend on this functionality are not portable.

To create a physical session, see [“SessionCreateTemporaryQueue” on page 406](#).

Parameters

Name	Type	Description
pSessn	SBYN_Session*	Pointer to the Session object.
pczQueName	char*	Pointer to the name of the queue.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

SBYN_Destination* pointer.

SessionCreateReceiver

Syntax

```
SessionCreateReceiver(pSessn, pDest, iError, pczError)
```

Description

Creates a **Receiver** object to receive messages;

Parameters

Name	Type	Description
pSessn	SBYN_Session*	Pointer to the Session object.
pDest	SBYN_Destination*	Pointer to the queue.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

SBYN_Receiver* pointer.

SessionCreateReceiveMessageSelector

Syntax

```
SessionCreateReceiveMessageSelector(pSessn, pDest,  
pczSelector, iError, pczError)
```

Description

Creates a **Receiver** object to receive messages using a message selector. Also see [“SessionCreateReceiver” on page 405](#).

Parameters

Name	Type	Description
pSessn	SBYN_Session*	Pointer to the object.
pDest	SBYN_Destination*	Pointer to the queue.
pczSelector	char*	Pointer to the text of the message selector.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

SBYN_QueueReceiver* pointer.

SessionCreateSender

Syntax

```
SessionCreateSender(pSessn, pDest, iError, pczError)
```

Description

Creates a **Sender** object to send messages.

Parameters

Name	Type	Description
pSessn	SBYN_Session*	Pointer to the Session object.
pDest	SBYN_Destination*	Pointer to the queue.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

SBYN_Sender* pointer.

SessionCreateTemporaryQueue

Syntax

```
SessionCreateTemporary(pSessn, iError, pczError)
```

Description

Creates a **Temporary** object for a specified session.

Parameters

Name	Type	Description
pSessn	SBYN_Session*	Pointer to the Session object.

Name	Type	Description
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

SBYN_Destination* pointer.

SessionCreateDurableSubscriber

Syntax

```
SessionCreateDurableSubscriber(pSessn, pDest, pczName,
                              iError, pczError)
```

Description

Creates a durable subscriber to the specified topic, specifying whether messages published by its own connection should be delivered to it.

Using pub/sub messaging, if a client needs to receive all the messages published on a topic, including messages published while the subscriber is inactive, it uses a *durable subscriber*. The JMS provider retains a record of this durable subscription and ensures that each message from the topic's publishers is retained until either it has been acknowledged by this durable subscriber or else it has expired.

Sessions with durable subscribers must always provide the same client ID, and each client must specify a name that (within the given client ID) uniquely identifies each durable subscription it creates. Only one session at a time can have a TopicSubscriber for a particular durable subscription. An *inactive* durable subscriber is one that exists but does not currently have a message consumer associated with it.

A client can change an existing durable subscription by creating a durable TopicSubscriber with the same name and a new topic (and/or *message selector*). Changing a durable subscriber is equivalent to deleting the old one and creating a new one.

Parameters

Name	Type	Description
pSessn	SBYN_Session*	Pointer to the Session object.
pDest	SBYN_Destination*	Pointer to the topic.
pczName	char*	Pointer to the text string containing the client ID of the durable subscriber.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

SBYN_TopicSubscriber*
Pointer to the created **TopicSubscriber** object.

SessionCreateDurableSubscriberMessageSelector

Syntax

```
TopicSessionCreateDurableSubscriberMessageSelector(pSessn, pDest,
                                                    pDest, pczName, pczSelector, iError, pczError)
```

Description

Creates a durable subscriber to the specified topic, using a message selector (*pczSelector*) and/or specifying whether messages published by its own connection should be delivered to it (*fNoLocal*).

Parameters

Name	Type	Description
<i>pSessn</i>	SBYN_Session*	Pointer to the Session object.
<i>pDest</i>	SBYN_Destination*	Pointer to the Destination object.
<i>pczName</i>	char*	Pointer to the text string containing the client ID of the durable subscriber.
<i>pczSelector</i>	char*	Pointer to the string containing the text of the message selector; only messages with properties matching the message selector expression are delivered. A value of null or an empty string indicates that there is no message selector for the message consumer.
<i>fNoLocal</i>	SBYN_BOOL	Flag, default false . If true , inhibits the delivery of messages published by its own connection.
<i>iError</i>	int*	Pointer to the error number.
<i>pczError</i>	char*	Pointer to the error message text.

Return Value

SBYN_TopicSubscriber*
Pointer to the created **TopicSubscriber** object.

SessionCreatePublisher

Syntax

```
TopicSessionCreatePublisher(pSessn, pDest, iError, pczError)
```

Description

Creates a publisher for the specified topic.

Parameters

Name	Type	Description
<i>pSessn</i>	SBYN_Session*	Pointer to the Session object.

Name	Type	Description
pDest	SBYN_Destination*	Pointer to the Destination object.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

SBYN_TopicPublisher*

Pointer to the created **TopicPublisher** object.

SessionCreateSubscriber

Syntax

```
SessionCreateSubscriber(pSessn, pDest, iError, pczError)
```

Description

Creates a *nondurable* subscriber to the specified topic—in other words, a subscriber that receives only those messages that are published while the subscriber is active.

Parameters

Name	Type	Description
pSessn	SBYN_Session*	Pointer to the Session object.
pDest	SBYN_Destination*	Pointer to the Destination object.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

SBYN_TopicSubscriber*

Pointer to the **TopicSubscriber** object.

SessionCreateSubscriberMessageSelector

Syntax

```
SessionCreateSubscriberMessageSelector(pSessn, pDest,
pczSelector, fNoLocal, iError, pczError)
```

Description

Creates a *nondurable* subscriber to the specified topic—in other words, a subscriber that receives only those messages that are published while the subscriber is active.

In some cases, a connection may both publish and subscribe to a topic. The `NoLocal` parameter allows a subscriber to inhibit the delivery of messages published by its own connection. The default value for this attribute is **false**.

Parameters

Name	Type	Description
pSessn	SBYN_Session*	Pointer to the Session object.
pDest	SBYN_Destination*	Pointer to the Destination object.
pczSelector	char*	Pointer to the string containing the text of the message selector; only messages with properties matching the message selector expression are delivered. A value of null or an empty string indicates that there is no message selector for the message consumer.
fNoLocal	SBYN_BOOL	Flag, default false . If true , inhibits the delivery of messages published by its own connection.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

SBYN_TopicSubscriber*
Pointer to the **TopicSubscriber** object.

SessionCreateTemporaryTopic

Syntax

```
SessionTopicSessionCreateTemporaryTopic (pSessn,
                                           iError, pczError)
```

Description

Creates a temporary topic that lives only as long as the specified TopicConnection does (unless the topic is deleted earlier).

Parameters

Name	Type	Description
pSessn	SBYN_Session*	Pointer to the Session object.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

SBYN_Destination*
Pointer to the created **TemporaryTopic** object.

SessionCreateTopic

Syntax

```
SessionCreateTopic(pSessn, pczTopName, iError, pczError)
```

Description

Creates a topic identity with a specific topic name; does *not* create a physical topic.

This functionality is provided for rare cases where clients need to dynamically create a topic identity with a provider-specific name. Clients that depend on this functionality are not portable.

Parameters

Name	Type	Description
pSessn	SBYN_Session*	Pointer to the Session object.
pczTopName	char*	Pointer to the text string containing the name of the topic.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

SBYN_Destination*

Pointer to the created **Destination** object.

SessionUnsubscribe

Syntax

```
SessionUnsubscribe(pSessn, pczName, iError, pczError)
```

Description

Unsubscribes a durable subscription that has been created by a client. Note that it is an error to delete a durable subscription while there is an active TopicSubscriber for the subscription, or while a consumed message is part of a pending transaction or has not been acknowledged in the session.

Parameters

Name	Type	Description
pSessn	SBYN_Session*	Pointer to the Session object.
pczName	char*	Pointer to the text string containing the name used to identify this subscription.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

None.

5.8.12. The TopicConnectionFactory Interface

Using pub/sub messaging, a client uses a **TopicConnectionFactory** object to create **TopicConnection** objects.

The **TopicConnectionFactory** interface includes the following methods:

- [CreateTopicConnectionFactory](#) on page 412
- [CreateTopicConnection](#) on page 412

CreateTopicConnectionFactory

Syntax

```
CreateTopicConnectionFactory(pczHost, iPort, iError, pczError)
```

Description

Constructs a **TopicConnectionFactory** for the specified host and port. Once constructed, it can create **TopicConnection** objects for a pub/sub JMS provider.

Parameters

Name	Type	Description
<i>pczHost</i>	char*	Pointer to the text of the host name.
<i>iPort</i>	int	Port number.
<i>iError</i>	int*	Pointer to the error number.
<i>pczError</i>	char*	Pointer to the error message text.

Return Value

SBYN_TopicConnectionFactory*

Pointer to the **TopicConnectionFactory** object that was created.

CreateTopicConnection

Syntax

```
CreateTopicConnection(pTopCxnFac, iError, pczError)
```

Description

Constructs a **TopicConnection** object.

Parameters

Name	Type	Description
pTopCxnFac	SBYN_TopicConnectionFactory*	Pointer to the TopicConnectionFactory object creating the TopicConnection.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

SBYN_TopicConnection*

Pointer to the **TopicConnection** object that was created.

5.8.13. The Destination Interface

A **Destination** object encapsulates an address for a destination provided by JMS. It can also include other data, such as configuration information or metadata.

The **Destination** interface includes the following methods:

- [GetDestinationName](#) on page 413
- [SetDestinationName](#) on page 414
- [DestinationToString](#) on page 414
- [DeleteDestination](#) on page 414

GetDestinationName

Syntax

```
GetDestinationName(pDest, iError, pczError)
```

Description

Retrieves the name of the specified **Destination** object.

Parameters

Name	Type	Description
pDest	SBYN_Destination*	Pointer to the Destination object.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

WString*

Pointer to a **WString** (wide string) object containing the text.

SetDestinationName

Syntax

```
SetDestinationName(pDest, pczName, iError, pczError)
```

Description

Sets the name of the specified **Destination** object.

Parameters

Name	Type	Description
<i>pDest</i>	SBYN_Destination*	Pointer to the Destination object.
<i>pczName</i>	char*	Pointer to the text string containing the name of the destination.
<i>iError</i>	int*	Pointer to the error number.
<i>pczError</i>	char*	Pointer to the error message text.

Return Value

None.

DestinationToString

Syntax

```
DestinationToString(pDest, iError, pczError)
```

Description

Retrieves a text string representation of the specified **Destination** object.

Parameters

Name	Type	Description
<i>pDest</i>	SBYN_Destination*	Pointer to the Destination object.
<i>iError</i>	int*	Pointer to the error number.
<i>pczError</i>	char*	Pointer to the error message text.

Return Value

WString*

Pointer to a **WString** (wide string) object containing the text.

DeleteDestination

Syntax

```
DeleteDestination(pDest, iError, pczError)
```

Description

Deletes the specified **Destination** object.

Parameters

Name	Type	Description
pDest	SBYN_Destination*	Pointer to the Destination object.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

None.

5.8.14. The QueueReceiver Interface

Using point-to-point messaging, a client uses a **QueueReceiver** object to receive messages that have been delivered to a queue.

The **QueueReceiver** interface includes the following methods:

- [QueueReceiverClose](#) on page 415
- [QueueReceiverGetMessageSelector](#) on page 416
- [QueueReceiverReceive](#) on page 416
- [QueueReceiverReceiveTimeout](#) on page 417
- [QueueReceiverReceiveNoWait](#) on page 417
- [QueueReceiverGetQueue](#) on page 417

QueueReceiverClose

Syntax

```
QueueReceiverClose(pQueueRecvr, iError, pczError)
```

Description

Closes the specified queue receiver.

Note: When a message consumer is no longer needed, it should be closed.

Parameters

Name	Type	Description
pQueueRecvr	SBYN_QueueReceiver*	Pointer to the QueueReceiver object.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

None.

QueueReceiverGetMessageSelector

Syntax

```
QueueReceiverGetMessageSelector(pQueRecvr, iError, pczError)
```

Description

Retrieves the message selector expression associated with this queue receiver.

Parameters

Name	Type	Description
<i>pQueRecvr</i>	SBYN_QueueReceiver*	Pointer to the QueueReceiver object.
<i>iError</i>	int*	Pointer to the error number.
<i>pczError</i>	char*	Pointer to the error message text.

Return Value

char*

Pointer to the string containing the message selector text. Returns null if no message selector exists, or if the message selector was set to null or the empty string.

QueueReceiverReceive

Syntax

```
QueueReceiverReceive(pQueRecvr, iError, pczError)
```

Description

Receives the next message produced for this queue. If the called within a transaction, the queue retains the message until the transaction commits.

Parameters

Name	Type	Description
<i>pQueRecvr</i>	SBYN_QueueReceiver*	Pointer to the QueueReceiver object.
<i>iError</i>	int*	Pointer to the error number.
<i>pczError</i>	char*	Pointer to the error message text.

Return Value

SBYN_Message*

Pointer to the received message.

QueueReceiverReceiveTimeout

Syntax

```
QueueReceiverReceiveTimeout(pQueRecvr, lTimeout, iError, pczError)
```

Description

Receives the next message that arrives within the specified timeout interval. A timeout value of 0 makes this function equivalent to [“QueueReceiverReceive” on page 416](#).

Parameters

Name	Type	Description
<i>pQueRecvr</i>	SBYN_QueueReceiver*	Pointer to the QueueReceiver object.
<i>lTimeout</i>	long	The timeout value.
<i>iError</i>	int*	Pointer to the error number.
<i>pczError</i>	char*	Pointer to the error message text.

Return Value

SBYN_Message*

Pointer to the received message, or null if no message is available.

QueueReceiverReceiveNoWait

Syntax

```
QueueReceiverReceiveNoWait(pQueRecvr, iError, pczError)
```

Description

Receives the next message produced for this queue if one is immediately available.

Parameters

Name	Type	Description
<i>pQueRecvr</i>	SBYN_QueueReceiver*	Pointer to the QueueReceiver object.
<i>iError</i>	int*	Pointer to the error number.
<i>pczError</i>	char*	Pointer to the error message text.

Return Value

SBYN_Message*

Pointer to the received message, or null if no message is available.

QueueReceiverGetQueue

Syntax

```
QueueReceiverGetQueue(pQueRecvr, iError, pczError)
```

Description

Retrieves the queue associated with the specified queue receiver.

Parameters

Name	Type	Description
pQueRecvr	SBYN_QueueReceiver*	Pointer to the QueueReceiver object.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

SBYN_Destination*

Pointer to the queue.

5.8.15. The TopicSubscriber Interface

The TopicSubscriber interface includes the following methods:

- [TopicSubscriberClose](#) on page 418
- [TopicSubscriberGetMessageSelector](#) on page 419
- [TopicSubscriberGetNoLocal](#) on page 419
- [TopicSubscriberGetTopic](#) on page 420
- [TopicSubscriberReceive](#) on page 420
- [TopicSubscriberReceiveTimeout](#) on page 420
- [TopicSubscriberReceiveNoWait](#) on page 421

TopicSubscriberClose

Syntax

```
TopicSubscriberClose(pTopSub, iError, pczError)
```

Description

Closes the specified topic subscriber.

Note: When a message consumer is no longer needed, it should be closed.

Parameters

Name	Type	Description
pTopSub	SBYN_TopicSubscriber*	Pointer to the TopicSubscriber object.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

None.

TopicSubscriberGetMessageSelector

Syntax

```
TopicSubscriberGetMessageSelector(pTopSub, iError, pczError)
```

Description

Retrieves the message selector expression associated with the specified topic subscriber.

Parameters

Name	Type	Description
<i>pTopSub</i>	SBYN_TopicSubscriber*	Pointer to the TopicSubscriber object.
<i>iError</i>	int*	Pointer to the error number.
<i>pczError</i>	char*	Pointer to the error message text.

Return Value

char*

Pointer to the string containing the message selector text. Returns null if no message selector exists, or if the message selector was set to null or the empty string.

TopicSubscriberGetNoLocal

Syntax

```
TopicSubscriberGetNoLocal(pTopSub, iError, pczError)
```

Description

Queries whether the NoLocal flag is set for the specified topic subscriber.

Parameters

Name	Type	Description
<i>pTopSub</i>	SBYN_TopicSubscriber*	Pointer to the TopicSubscriber object.
<i>iError</i>	int*	Pointer to the error number.
<i>pczError</i>	char*	Pointer to the error message text.

Return Value

SBYN_BOOL

If **false** (the default), the subscriber can also publish via the same connection.
If **true**, delivery of messages published via its own connection is inhibited.

TopicSubscriberGetTopic

Syntax

```
TopicSubscriberGetTopic(pTopSub, iError, pczError)
```

Description

Retrieves the topic associated with the specified topic subscriber.

Parameters

Name	Type	Description
<i>pTopSub</i>	SBYN_TopicSubscriber*	Pointer to the TopicSubscriber object.
<i>iError</i>	int*	Pointer to the error number.
<i>pczError</i>	char*	Pointer to the error message text.

Return Value

SBYN_Destination*
Pointer to the topic.

TopicSubscriberReceive

Syntax

```
TopicSubscriberReceive(pTopSub, iError, pczError)
```

Description

Receives the next message produced for this topic. If called within a transaction, the topic retains the message until the transaction commits.

Parameters

Name	Type	Description
<i>pTopSub</i>	SBYN_TopicSubscriber*	Pointer to the TopicSubscriber object.
<i>iError</i>	int*	Pointer to the error number.
<i>pczError</i>	char*	Pointer to the error message text.

Return Value

SBYN_Message*
Pointer to the received message.

TopicSubscriberReceiveTimeout

Syntax

```
TopicSubscriberReceiveTimeout(pTopSub, lTimeout, iError, pczError)
```

Description

Receives the next message that arrives within the specified timeout interval. A timeout value of 0 makes this function equivalent to “[TopicSubscriberReceive](#)” on page 420.

Parameters

Name	Type	Description
pTopSub	SBYN_TopicSubscriber*	Pointer to the TopicSubscriber object.
lTimeout	long	The timeout value
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

SBYN_Message*

Pointer to the received message, or null if no message is available.

TopicSubscriberReceiveNoWait

Syntax

```
TopicSubscriberReceiveNoWait(pTopSub, iError, pczError)
```

Description

Receives the next message produced for this topic if one is immediately available.

Parameters

Name	Type	Description
pTopSub	SBYN_TopicSubscriber*	Pointer to the TopicSubscriber object.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

SBYN_Message*

Pointer to the received message, or null if no message is available.

Syntax

```
TopicSubscriberReceiveNoWait(pTopSub, iError, pczError)
```

5.8.16. The QueueSender Interface

Using point-to-point messaging, a client uses a **QueueSender** object to send messages to a queue. After sending a message to a queue, a client may retain the message and modify it without affecting the message that has been sent. The same message object may be sent multiple times.

The **QueueSender** interface includes the following methods:

- [QueueSenderClose](#) on page 422
- [QueueSenderGetDeliveryMode](#) on page 423
- [QueueSenderGetDisableMessageID](#) on page 423
- [QueueSenderGetDisableMessageTimestamp](#) on page 424
- [QueueSenderGetJMS_ProducerID](#) on page 424
- [QueueSenderGetPriority](#) on page 424
- [QueueSenderGetQueue](#) on page 425
- [QueueSenderGetTimeToLive](#) on page 425
- [QueueSenderSend](#) on page 426
- [QueueSenderSendEx](#) on page 426
- [QueueSenderSendToQueue](#) on page 427
- [QueueSenderSendToQueueEx](#) on page 428
- [QueueSenderSetDeliveryMode](#) on page 429
- [QueueSenderSetDisableMessageID](#) on page 429
- [QueueSenderSetDisableMessageTimestamp](#) on page 430
- [QueueSenderSetJMS_ProducerID](#) on page 430
- [QueueSenderSetPriority](#) on page 431
- [QueueSenderSetTimeToLive](#) on page 431

QueueSenderClose

Syntax

```
QueueSenderClose(pQueueSender, iError, pczError)
```

Description

Closes the specified queue sender.

Note: When a message producer is no longer needed, it should be closed.

Parameters

Name	Type	Description
pQueueSender	SBYN_QueueSender*	Pointer to the QueueSender object.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

None.

QueueSenderGetDeliveryMode

Syntax

```
QueueSenderGetDeliveryMode(pQueueSender, iError, pczError)
```

Description

Retrieves the value of the **DeliveryMode** property of the specified queue sender.

Parameters

Name	Type	Description
pQueueSender	SBYN_QueueSender*	Pointer to the QueueSender object.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

int

See [“DeliveryMode Constants” on page 358](#).

QueueSenderGetDisableMessageID

Syntax

```
QueueSenderGetDisableMessageID(pQueueSender, iError, pczError)
```

Description

Queries whether message IDs are or are not disabled for the specified queue sender.

Parameters

Name	Type	Description
pQueueSender	SBYN_QueueSender*	Pointer to the QueueSender object.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

SBYN_BOOL

Returns **true** if message IDs are disabled; otherwise, returns **false**.

QueueSenderGetDisableMessageTimestamp

Syntax

```
QueueSenderGetDisableMessageTimestamp(pQueueSender, iError, pczError)
```

Description

Queries whether message timestamping is or is not disabled for the specified queue sender.

Parameters

Name	Type	Description
<i>pQueueSender</i>	SBYN_QueueSender*	Pointer to the QueueSender object.
<i>iError</i>	int*	Pointer to the error number.
<i>pczError</i>	char*	Pointer to the error message text.

Return Value

SBYN_BOOL

Returns **true** if message timestamping is disabled; otherwise, returns **false**.

QueueSenderGetJMS_ProducerID

Syntax

```
QueueSenderGetJMS_ProducerID(pQueueSender, iError, pczError)
```

Description

Retrieves the value of the **ProducerID** property for the specified queue sender.

Parameters

Name	Type	Description
<i>pQueueSender</i>	SBYN_QueueSender*	Pointer to the QueueSender object.
<i>iError</i>	int*	Pointer to the error number.
<i>pczError</i>	char*	Pointer to the error message text.

Return Value

WString*

Pointer to a **WString** (wide string) object containing the text.

QueueSenderGetPriority

Syntax

```
QueueSenderGetPriority(pQueueSender, iError, pczError)
```


Description

Queries the value of the message **Priority** property of the specified queue sender.

Parameters

Name	Type	Description
pQueueSender	SBYN_QueueSender*	Pointer to the QueueSender object.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

int

Message priority level, from **0** (least expedited) through **9** (most expedited). See [“Miscellaneous Constants Setting Message Class Defaults” on page 360](#).

QueueSenderGetQueue

Syntax

```
QueueSenderGetQueue(pQueueSender, iError, pczError)
```

Description

Retrieves the queue associated with the queue sender.

Parameters

Name	Type	Description
pQueueSender	SBYN_QueueSender*	Pointer to the QueueSender object.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

SBYN_Destination*

Pointer to the queue.

QueueSenderGetTimeToLive

Syntax

```
QueueSenderGetTimeToLive(pQueueSender, iError, pczError)
```

Description

Queries the value of the **TimeToLive** property of the specified queue sender.

Parameters

Name	Type	Description
pQueueSender	SBYN_QueueSender*	Pointer to the QueueSender object.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

long

Length of time (milliseconds elapsed since dispatch time) that a produced message should be retained by JMS. See [“Miscellaneous Constants Setting Message Class Defaults” on page 360](#).

QueueSenderSend

Syntax

```
QueueSenderSend(pQueueSender, pMsg, iError, pczError)
```

Description

Sends the specified message to the queue, using default values for properties (**DeliveryMode**, **Priority**, and **TimeToLive**) of the specified queue sender. If you need to override the default values, see [“QueueSenderSendEx” on page 426](#).

To specify the queue destination, see [“QueueSenderSendToQueue” on page 427](#).

Parameters

Name	Type	Description
pQueueSender	SBYN_QueueSender*	Pointer to the QueueSender object.
pMsg	SBYN_Message*	Pointer to the message to send.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

None.

QueueSenderSendEx

Syntax

```
QueueSenderSendEx(pQueueSender, pMsg,  
                  iDelivMode, iPriority, lMilSecToLive,  
                  iError, pczError)
```

Description

Sends the specified message to the queue, overriding one or more default values for properties of the specified queue sender. Compare to [“QueueSenderSend” on page 426](#).

To specify the queue destination, see [“QueueSenderSendToQueueEx” on page 428](#).

Parameters

Name	Type	Description
pQueueSender	SBYN_QueueSender*	Pointer to the QueueSender object.
pMsg	SBYN_Message*	Pointer to the message to send.
iDelivMode	int	Value to be used for DeliveryMode ; see “DeliveryMode Constants” on page 358 .
iPriority	int	Value to be used for Priority ; see “Miscellaneous Constants Setting Message Class Defaults” on page 360 .
IMilSecToLive	long	Value to be used for TimeToLive : Length of time (milliseconds elapsed since dispatch time) that a produced message should be retained by JMS. See “Miscellaneous Constants Setting Message Class Defaults” on page 360 .
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

None.

QueueSenderSendToQueue

Syntax

```
QueueSenderSendToQueue(pQueueSender, pDest, pMsg, iError, pczError)
```

Description

Sends the specified message to the specified queue, using default values for properties (**DeliveryMode**, **Priority**, and **TimeToLive**) of the specified queue sender. Compare to [“QueueSenderSendToQueueEx” on page 428](#).

Typically, a message producer is assigned a queue at creation time; however, the JMS API also supports unidentified message producers, which require that the queue be supplied every time a message is sent. Compare to [“QueueSenderSend” on page 426](#).

Parameters

Name	Type	Description
pQueueSender	SBYN_QueueSender*	Pointer to the QueueSender object.

Name	Type	Description
pDest	SBYN_Destination*	Pointer to the queue.
pMsg	SBYN_Message*	Pointer to the message to send.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

None.

QueueSenderSendToQueueEx

Syntax

```
QueueSenderSendToQueueEx(pQueueSender, pDest, pMsg,
                          iDelivMode, iPriority, lMilSecToLive,
                          iError, pczError)
```

Description

Sends the specified message to the specified queue, overriding one or more default values for properties of the specified queue sender. Compare to [“QueueSenderSendToQueue” on page 427](#).

Parameters

Name	Type	Description
pQueueSender	SBYN_QueueSender*	Pointer to the QueueSender object.
pDest	SBYN_Destination*	Pointer to the queue.
pMsg	SBYN_Message*	Pointer to the message to send.
iDelivMode	int	Value to be used for DeliveryMode ; see “DeliveryMode Constants” on page 358 .
iPriority	int	Value to be used for Priority ; see “Miscellaneous Constants Setting Message Class Defaults” on page 360 .
lMilSecToLive	long	Value to be used for TimeToLive : Length of time (milliseconds elapsed since dispatch time) that a produced message should be retained by JMS. See “Miscellaneous Constants Setting Message Class Defaults” on page 360 .
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

None.

QueueSenderSetDeliveryMode

Syntax

```
QueueSenderSetDeliveryMode(pQueueSender, iDelivMode, iError, pczError)
```

Description

Sets the value of the **DeliveryMode** property of the specified queue sender.
See [“DeliveryMode Constants” on page 358](#).

Parameters

Name	Type	Description
<i>pQueueSender</i>	SBYN_QueueSender*	Pointer to the QueueSender object.
<i>iDelivMode</i>	int	Value for the DeliveryMode property.
<i>iError</i>	int*	Pointer to the error number.
<i>pczError</i>	char*	Pointer to the error message text.

Return Value

None.

QueueSenderSetDisableMessageID

Syntax

```
QueueSenderSetDisableMessageID(pQueueSender, fDisabled,  
                                iError, pczError)
```

Description

Determines whether message IDs are disabled for this queue sender. Default **false**.

Parameters

Name	Type	Description
<i>pQueueSender</i>	SBYN_QueueSender*	Pointer to the QueueSender object.
<i>fDisabled</i>	SBYN_BOOL	Flag, default false ; if true , message IDs are disabled.
<i>iError</i>	int*	Pointer to the error number.
<i>pczError</i>	char*	Pointer to the error message text.

Return Value

None.

QueueSenderSetDisableMessageTimestamp

Syntax

```
QueueSenderSetDisableMessageTimestamp(pQueueSender, fDisabled,  
                                       iError, pczError)
```

Description

Determines whether message timestamping is disabled for this queue sender.
Default **false**.

Since message timestamps take effort to create and increase the size of a message, this flag can be set **true** to reduce overhead if message IDs are not used by an application.

Parameters

Name	Type	Description
<i>pQueueSender</i>	SBYN_QueueSender*	Pointer to the QueueSender object.
<i>fDisabled</i>	SBYN_BOOL	Flag, default false ; if true , message timestamping is disabled.
<i>iError</i>	int*	Pointer to the error number.
<i>pczError</i>	char*	Pointer to the error message text.

Return Value

None.

QueueSenderSetJMS_ProducerID

Syntax

```
QueueSenderSetJMS_ProducerID(pQueueSender, pczProdID, iError, pczError)
```

Description

Sets the value of the **ProducerID** property for the specified queue sender.

Parameters

Name	Type	Description
<i>pQueueSender</i>	SBYN_QueueSender*	Pointer to the QueueSender object.
<i>pczProdID</i>	char*	Pointer to text string containing the Producer ID.
<i>iError</i>	int*	Pointer to the error number.
<i>pczError</i>	char*	Pointer to the error message text.

Return Value

None.

QueueSenderSetPriority

Syntax

```
QueueSenderSetPriority(pQueueSender, iPriority, iError, pczError)
```

Description

Sets the value of the message **Priority** property, from **0** (least expedited) through **9** (most expedited). See [“Miscellaneous Constants Setting Message Class Defaults” on page 360](#).

Parameters

Name	Type	Description
pQueueSender	SBYN_QueueSender*	Pointer to the QueueSender object.
iPriority	int	Message priority level.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

None.

QueueSenderSetTimeToLive

Syntax

```
QueueSenderSetTimeToLive(pQueueSender, lMilSecToLive, iError, pczError)
```

Description

Sets the value of the **TimeToLive** property of the specified queue sender.

Parameters

Name	Type	Description
pQueueSender	SBYN_QueueSender*	Pointer to the QueueSender object.
lMilSecToLive	long	Value to be used for TimeToLive: Length of time (milliseconds elapsed since dispatch time) that a produced message should be retained by JMS. See “Miscellaneous Constants Setting Message Class Defaults” on page 360 .
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

None.

5.8.17. The TopicPublisher Interface

The TopicPublisher interface includes the following methods:

- [TopicPublisherClose](#) on page 432
- [TopicPublisherGetDeliveryMode](#) on page 433
- [TopicPublisherGetDisableMessageID](#) on page 433
- [TopicPublisherGetDisableMessageTimestamp](#) on page 434
- [TopicPublisherGetJMS_ProducerID](#) on page 434
- [TopicPublisherGetPriority](#) on page 434
- [TopicPublisherGetTimeToLive](#) on page 435
- [TopicPublisherGetTopic](#) on page 435
- [TopicPublisherPublish](#) on page 436
- [TopicPublisherPublishEx](#) on page 436
- [TopicPublisherPublishToTopic](#) on page 437
- [TopicPublisherPublishToTopicEx](#) on page 438
- [TopicPublisherSetDeliveryMode](#) on page 438
- [TopicPublisherSetDisableMessageID](#) on page 439
- [TopicPublisherSetDisableMessageTimestamp](#) on page 439
- [TopicPublisherSetJMS_ProducerID](#) on page 440
- [TopicPublisherSetPriority](#) on page 440
- [TopicPublisherSetTimeToLive](#) on page 441

TopicPublisherClose

Syntax

```
TopicPublisherClose(pTopPub, iError, pczError)
```

Description

Closes the specified topic publisher.

Note: When a message producer is no longer needed, it should be closed.

Parameters

Name	Type	Description
pTopPub	SBYN_TopicPublisher*	Pointer to the TopicPublisher object.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

None.

TopicPublisherGetDeliveryMode

Syntax

```
TopicPublisherGetDeliveryMode(pTopPub, iError, pczError)
```

Description

Retrieves the value of the **DeliveryMode** property of the specified topic publisher.

Parameters

Name	Type	Description
pTopPub	SBYN_TopicPublisher*	Pointer to the TopicPublisher object.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

int

See [“DeliveryMode Constants” on page 358](#).

TopicPublisherGetDisableMessageID

Syntax

```
TopicPublisherGetDisableMessageID(pTopPub, iError, pczError)
```

Description

Queries whether message IDs are or are not disabled for the specified topic publisher.

Parameters

Name	Type	Description
pTopPub	SBYN_TopicPublisher*	Pointer to the TopicPublisher object.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

SBYN_BOOL

Returns **true** if message IDs are disabled; otherwise, returns **false**.

TopicPublisherGetDisableMessageTimestamp

Syntax

```
TopicPublisherGetDisableMessageTimestamp(pTopPub, iError, pczError)
```

Description

Queries whether message IDs are or are not disabled for the specified topic publisher.

Parameters

Name	Type	Description
<i>pTopPub</i>	SBYN_TopicPublisher*	Pointer to the TopicPublisher object.
<i>iError</i>	int*	Pointer to the error number.
<i>pczError</i>	char*	Pointer to the error message text.

Return Value

int

Returns **true** if message timestamping is disabled; otherwise, returns **false**.

TopicPublisherGetJMS_ProducerID

Syntax

```
TopicPublisherGetJMS_ProducerID(pTopPub, iError, pczError)
```

Description

Retrieves the value of the **ProducerID** property for the specified topic publisher.

Parameters

Name	Type	Description
<i>pTopPub</i>	SBYN_TopicPublisher*	Pointer to the TopicPublisher object.
<i>iError</i>	int*	Pointer to the error number.
<i>pczError</i>	char*	Pointer to the error message text.

Return Value

WString*

Pointer to a **WString** (wide string) object containing the text.

TopicPublisherGetPriority

Syntax

```
TopicPublisherGetPriority(pTopPub, iError, pczError)
```

Description

Queries the value of the message **Priority** property of the specified topic publisher.

Parameters

Name	Type	Description
pTopPub	SBYN_TopicPublisher*	Pointer to the TopicPublisher object.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

int

Message priority level, from **0** (least expedited) through **9** (most expedited). See [“Miscellaneous Constants Setting Message Class Defaults” on page 360](#).

TopicPublisherGetTimeToLive

Syntax

```
TopicPublisherGetTimeToLive(pTopPub, iError, pczError)
```

Description

Queries the value of the **TimeToLive** property of the specified topic publisher.

Parameters

Name	Type	Description
pTopPub	SBYN_TopicPublisher*	Pointer to the TopicPublisher object.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

long

Length of time (milliseconds elapsed since dispatch time) that a produced message should be retained by JMS. See [“Miscellaneous Constants Setting Message Class Defaults” on page 360](#).

TopicPublisherGetTopic

Syntax

```
TopicPublisherGetTopic(pTopPub, iError, pczError)
```

Description

Retrieves the topic associated with the specified topic publisher.

Parameters

Name	Type	Description
pTopPub	SBYN_TopicPublisher*	Pointer to the TopicPublisher object.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

SBYN_Destination*
Pointer to the topic.

TopicPublisherPublish

Syntax

```
TopicPublisherPublish(pTopPub, pMsg, iError, pczError)
```

Description

Publishes the specified message to the topic, using default values for properties (**DeliveryMode**, **Priority**, and **TimeToLive**) of the specified topic publisher. If you need to override the default values, see [“TopicPublisherPublishEx” on page 436](#).

To specify the topic destination, see [“TopicPublisherPublishToTopic” on page 437](#).

Parameters

Name	Type	Description
pTopPub	SBYN_TopicPublisher*	Pointer to the TopicPublisher object.
pMsg	SBYN_Message*	Pointer to the message to be published.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

None.

TopicPublisherPublishEx

Syntax

```
TopicPublisherPublishEx(pTopPub, pMsg,
                        iDelivMode, iPriority, lMilSecToLive,
                        iError, pczError)
```

Description

Publishes the specified message to the topic, overriding one or more default values for properties of the specified topic publisher. Compare to [“TopicPublisherPublish” on page 436](#).

To specify the topic destination, see [“TopicPublisherPublishToTopicEx” on page 438](#).

Parameters

Name	Type	Description
pTopPub	SBYN_TopicPublisher*	Pointer to the TopicPublisher object.
pMsg	SBYN_Message*	Pointer to the message to be published.
iDelivMode	int	Value to be used for DeliveryMode ; see “DeliveryMode Constants” on page 358 .
iPriority	int	Value to be used for Priority ; see “Miscellaneous Constants Setting Message Class Defaults” on page 360 .
lMilSecToLive	long	Value to be used for TimeToLive : Length of time (milliseconds elapsed since dispatch time) that a produced message should be retained by JMS. See “Miscellaneous Constants Setting Message Class Defaults” on page 360 .
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

None.

TopicPublisherPublishToTopic

Syntax

```
TopicPublisherPublishToTopic(pTopPub, pDest, pMsg, iError, pczError)
```

Description

Publishes the specified message to the specified topic, using default values for properties (**DeliveryMode**, **Priority**, and **TimeToLive**) of the specified topic publisher. Compare to [“TopicPublisherPublishToTopicEx” on page 438](#).

Typically, a message producer is assigned a topic at creation time; however, the JMS API also supports unidentified message producers, which require that the topic be supplied every time a message is sent. Compare to [“TopicPublisherPublishEx” on page 436](#).

Parameters

Name	Type	Description
pTopPub	SBYN_TopicPublisher*	Pointer to the TopicPublisher object.
pDest	SBYN_Destination*	Pointer to the topic.
pMsg	SBYN_Message*	Pointer to the message.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

None.

TopicPublisherPublishToTopicEx

Syntax

```
TopicPublisherPublishToTopicEx(pTopPub, pDest, pMsg,
                               iDelivMode, iPriority, lMilSecToLive,
                               iError, pczError)
```

Description

Publishes the specified message to the specified topic, overriding one or more default values for properties of the specified topic publisher. Compare to [“TopicPublisherPublishToTopic” on page 437](#).

Parameters

Name	Type	Description
pTopPub	SBYN_TopicPublisher*	Pointer to the TopicPublisher object.
pDest	SBYN_Destination*	Pointer to the topic.
pMsg	SBYN_Message*	Pointer to the message to publish.
iDelivMode	int	Value to be used for DeliveryMode ; see “DeliveryMode Constants” on page 358 .
iPriority	int	Value to be used for Priority ; see “Miscellaneous Constants Setting Message Class Defaults” on page 360 .
lMilSecToLive	long	Value to be used for TimeToLive : Length of time (milliseconds elapsed since dispatch time) that a produced message should be retained by JMS. See “Miscellaneous Constants Setting Message Class Defaults” on page 360 .
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

None.

TopicPublisherSetDeliveryMode

Syntax

```
TopicPublisherSetDeliveryMode(pTopPub, iDelivMode, iError, pczError)
```

Description

Sets the value of the **DeliveryMode** property of the specified topic publisher. See [“DeliveryMode Constants” on page 358](#).

Parameters

Name	Type	Description
pTopPub	SBYN_TopicPublisher*	Pointer to the TopicPublisher object.
iDelivMode	int	Value for the DeliveryMode property.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

None.

TopicPublisherSetDisableMessageID

Syntax

```
TopicPublisherSetDisableMessageID(pTopPub, fDisabled,
                                  iError, pczError)
```

Description

Determines whether to disable message IDs for this topic publisher. Default **false**.

Parameters

Name	Type	Description
pTopPub	SBYN_TopicPublisher*	Pointer to the TopicPublisher object.
iDisable	SBYN_BOOL	Flag, default false ; if true , message IDs are disabled.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

None.

TopicPublisherSetDisableMessageTimestamp

Syntax

```
TopicPublisherSetDisableMessageTimestamp(pTopPub, fDisabled,
                                          iError, pczError)
```

Description

Determines whether message timestamping is disabled for this topic publisher. Default **false**.

Since message timestamps take effort to create and increase the size of a message, this flag can be set **true** to reduce overhead if message IDs are not used by an application.

Parameters

Name	Type	Description
pTopPub	SBYN_TopicPublisher*	Pointer to the TopicPublisher object.
fDisabled	SBYN_BOOL	Flag, default false ; if true , message timestamping is disabled.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

None.

TopicPublisherSetJMS_ProducerID

Syntax

```
TopicPublisherSetJMS_ProducerID(pTopPub, pczProdID, iError, pczError)
```

Description

Sets the value of the **ProducerID** property for the specified topic publisher.

Parameters

Name	Type	Description
pTopPub	SBYN_TopicPublisher*	Pointer to the TopicPublisher object.
pczProdID	char*	Pointer to the text string containing the Producer ID.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

None.

TopicPublisherSetPriority

Syntax

```
TopicPublisherSetPriority(pTopPub, iPriority09, iError, pczError)
```

Description

Sets the value of the message **Priority** property of the specified topic publisher.

Parameters

Name	Type	Description
pTopPub	SBYN_TopicPublisher*	Pointer to the TopicPublisher object.
iPriority09	int	Message priority, from 0 (least expedited) through 9 (most expedited). See “Miscellaneous Constants Setting Message Class Defaults” on page 360 .
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

None.

TopicPublisherSetTimeToLive

Syntax

```
TopicPublisherSetTimeToLive(pTopPub, lMSecToLive, iError, pczError)
```

Description

Sets the value of the **TimeToLive** property for the specified topic publisher.

Parameters

Name	Type	Description
pTopPub	SBYN_TopicPublisher*	Pointer to the TopicPublisher object.
lMlSecToLive	long	Length of time (milliseconds elapsed since dispatch time) that a produced message should be retained by JMS. See “Miscellaneous Constants Setting Message Class Defaults” on page 360 .
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

None.

5.8.18. The TopicRequestor Interface

The **TopicRequestor** object is used in request/reply messaging to simplify making service requests. Given a non-transacted **TopicSession** object and a destination **Topic**, it creates a **TemporaryTopic** object for the responses and provides a request method that sends the request message and waits for its reply.

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

- [CreateTopicRequestor](#) on page 442
- [TopicRequestorRequest](#) on page 442
- [TopicRequestorRequestTimeout](#) on page 443
- [TopicRequestorClose](#) on page 443

CreateTopicRequestor

Syntax

```
CreateTopicRequestor(pTopSessn, pDest, iError, pczError)
```

Description

Constructs a **TopicRequestor** object. This implementation assumes that the parent topic session is non-transacted and that the value of its **DeliveryMode** property is either **AUTO_ACKNOWLEDGE** or **DUPS_OK_ACKNOWLEDGE**.

Parameters

Name	Type	Description
pTopSessn	SBYN_TopicSession*	Pointer to the parent TopicSession object.
pDest	SBYN_Destination*	Pointer to the destination topic.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

SBYN_TopicRequestor*

Pointer to the constructed **TopicRequestor** object.

TopicRequestorRequest

Syntax

```
TopicRequestorRequest(pTopReq, pMsg, iError, pczError)
```

Description

Sends a request and waits for a reply, without any upper limit on the time to wait (compare [“TopicRequestorRequestTimeout” on page 443](#)). The temporary topic is used for the **JMSReplyTo** destination; the first reply is returned, and any following replies are discarded.

Parameters

Name	Type	Description
pTopReq	SBYN_TopicRequestor*	Pointer to the TopicRequestor object.
pMsg	SBYN_Message*	Pointer to the message.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

SBYN_Message*
Pointer to the reply message.

TopicRequestorRequestTimeout

Syntax

```
TopicRequestorRequestTimeout(pTopReq, pMsg, lTimeout,  
                             iError, pczError)
```

Description

Sends a request and waits for a reply, but only during the specified period of time.
Compare to [“TopicRequestorRequest” on page 442](#).

Parameters

Name	Type	Description
pTopReq	SBYN_TopicRequestor*	Pointer to the TopicRequestor object.
pMsg	SBYN_Message*	Pointer to the message.
lTimeout	long	Timeout value, in milliseconds.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

SBYN_Message*
Pointer to the reply message, or **0** if no reply arrives before the waiting period ends.

TopicRequestorClose

Syntax

```
TopicRequestorClose(pTopReq, iError, pczError)
```

Description

Closes the specified topic requestor.

Note: When a message producer or consumer is no longer needed, it should be closed.

Parameters

Name	Type	Description
pTopReq	SBYN_TopicRequestor*	Pointer to the TopicRequestor object.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

None.

5.8.19. The QueueRequestor Interface

The **QueueRequestor** object is used in request/reply messaging to simplify making service requests. Given a non-transacted **QueueSession** object and a destination **Queue**, it creates a **TemporaryQueue** object for the responses and provides a request method that sends the request message and waits for its reply.

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

- [CreateQueueRequestor](#) on page 444
- [QueueRequestorClose](#) on page 445
- [QueueRequestorRequest](#) on page 445
- [QueueRequestorRequestTimeout](#) on page 446

CreateQueueRequestor

Syntax

```
CreateQueueRequestor(pQueSessn, pDest, iError, pczError)
```

Description

Constructs a **QueueRequestor** object. This implementation assumes that the parent queue session is non-transacted and that the value of its **DeliveryMode** property is either **AUTO_ACKNOWLEDGE** or **DUPS_OK_ACKNOWLEDGE**.

Parameters

Name	Type	Description
pQueSessn	SBYN_QueueSession*	Pointer to the parent QueueSession object.
pDest	SBYN_Destination*	Pointer to the destination queue.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

SBYN_QueueRequestor*

Pointer to the constructed **QueueRequestor** object.

QueueRequestorClose

Syntax

```
QueueRequestorClose(pQueReq, iError, pczError)
```

Description

Closes the specified queue requestor.

Note: When a message producer or consumer is no longer needed, it should be closed.

Parameters

Name	Type	Description
pQueReq	SBYN_QueueRequestor*	Pointer to the QueueRequestor object.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

None.

QueueRequestorRequest

Syntax

```
QueueRequestorRequest(pQueReq, pMsg, iError, pczError)
```

Description

Sends a request and waits for a reply, without any upper limit on the time to wait (compare [“QueueRequestorRequestTimeout” on page 446](#)). The temporary queue is used for the **JMSReplyTo** destination. Only one reply per request is expected.

Parameters

Name	Type	Description
pTopReq	SBYN_TopicRequestor*	Pointer to the TopicRequestor object.
pMsg	SBYN_Message*	Pointer to the message.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

SBYN_Message*
Pointer to the reply message.

QueueRequestorRequestTimeout

Syntax

```
QueueRequestorRequestTimeout(pQueReq, pMsg, lTimeout,  
                             iError, pczError)
```

Description

Sends a request and waits for a reply, but only during the specified period of time. Compare to [“QueueRequestorRequest” on page 445](#).

Parameters

Name	Type	Description
pQueReq	SBYN_QueueRequestor*	Pointer to the QueueRequestor object.
pMsg	SBYN_Message*	Pointer to the message.
lTimeout	long	Timeout value, in milliseconds.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

SBYN_Message*
Pointer to the reply message, or **0** if no reply arrives before the waiting period ends.

5.8.20. Destructor Methods

Destructor methods include the following:

- [DeleteQueueConnectionFactory](#) on page 447
- [DeleteQueueConnection](#) on page 447
- [DeleteQueueReceiver](#) on page 448
- [DeleteQueueSender](#) on page 448
- [DeleteQueueSession](#) on page 449
- [DeleteTopicConnectionFactory](#) on page 449
- [DeleteTopicConnection](#) on page 450
- [DeleteTopicSession](#) on page 450
- [DeleteTopicSubscriber](#) on page 450
- [DeleteTopicRequestor](#) on page 451
- [DeleteQueueRequestor](#) on page 448

- [DeleteTopicPublisher](#) on page 451
- [DeleteMessage](#) on page 452
- [DeleteWString](#) on page 453
- [DeleteWStringList](#) on page 454

DeleteQueueConnectionFactory

Syntax

```
DeleteQueueConnectionFactory(pQueCxnFac, iError, pczError)
```

Description

Deletes the specified queue connection factory.

Parameters

Name	Type	Description
<i>pQueCxnFac</i>	SBYN_QueueConnectionFactory*	Pointer to the QueueConnectionFactory object.
<i>iError</i>	int*	Pointer to the error number.
<i>pczError</i>	char*	Pointer to the error message text.

Return Value

None.

DeleteQueueConnection

Syntax

```
DeleteQueueConnection(pQueCxn, iError, pczError)
```

Description

Deletes the specified queue connection.

Parameters

Name	Type	Description
<i>pQueCxn</i>	SBYN_QueueConnection*	Pointer to the QueueConnection object.
<i>iError</i>	int*	Pointer to the error number.
<i>pczError</i>	char*	Pointer to the error message text.

Return Value

None.

DeleteQueueReceiver

Syntax

```
DeleteQueueReceiver(pQueRecvr, iError, pczError)
```

Description

Deletes the specified queue receiver.

Parameters

Name	Type	Description
<i>pQueRecvr</i>	SBYN_QueueReceiver*	Pointer to the QueueReceiver object.
<i>iError</i>	int*	Pointer to the error number.
<i>pczError</i>	char*	Pointer to the error message text.

Return Value

None.

DeleteQueueRequestor

Syntax

```
DeleteQueueRequestor(pQueReq, iError, pczError)
```

Description

Deletes the specified queue requestor.

Parameters

Name	Type	Description
<i>pQueReq</i>	SBYN_QueueRequestor*	Pointer to the QueueRequestor object.
<i>iError</i>	int*	Pointer to the error number.
<i>pczError</i>	char*	Pointer to the error message text.

Return Value

None.

DeleteQueueSender

Syntax

```
DeleteQueueSender(pQueSender, iError, pczError)
```

Description

Deletes the specified queue sender.

Parameters

Name	Type	Description
pQueueSender	SBYN_QueueSender*	Pointer to the QueueSender object.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

None.

DeleteQueueSession

Syntax

```
DeleteQueueSession(pQueueSessn, iError, pczError)
```

Description

Deletes the specified queue session.

Parameters

Name	Type	Description
pQueueSessn	SBYN_QueueSession*	Pointer to the QueueSession object.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

None.

DeleteTopicConnectionFactory

Syntax

```
DeleteTopicConnectionFactory(pTopCxnFac, iError, pczError)
```

Description

Deletes the specified topic connection factory.

Parameters

Name	Type	Description
pTopCxnFac	SBYN_TopicConnectionFactory*	Pointer to the TopicConnectionFactory object.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

None.

DeleteTopicConnection

Syntax

```
DeleteTopicConnection(pTopCxn, iError, pczError)
```

Description

Deletes the specified topic connection.

Parameters

Name	Type	Description
<i>pTopCxn</i>	SBYN_TopicConnection*	Pointer to the TopicConnection object.
<i>iError</i>	int*	Pointer to the error number.
<i>pczError</i>	char*	Pointer to the error message text.

Return Value

None.

DeleteTopicSession

Syntax

```
DeleteTopicSession(pTopSession, iError, pczError)
```

Description

Deletes the specified topic session.

Parameters

Name	Type	Description
<i>pTopSessn</i>	SBYN_TopicSession*	Pointer to the TopicSession object.
<i>iError</i>	int*	Pointer to the error number.
<i>pczError</i>	char*	Pointer to the error message text.

Return Value

None.

DeleteTopicSubscriber

Syntax

```
DeleteTopicSubscriber(pTopSub, iError, pczError)
```

Description

Deletes the specified topic subscriber.

Parameters

Name	Type	Description
pTopSub	SBYN_TopicSubscriber*	Pointer to the TopicSubscriber object.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

None.

DeleteTopicRequestor

Syntax

```
DeleteTopicRequestor(pTopReq, iError, pczError)
```

Description

Deletes the specified topic requestor.

Parameters

Name	Type	Description
qTopReq	SBYN_TopicRequestor*	Pointer to the TopicRequestor object.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

None.

DeleteTopicPublisher

Syntax

```
DeleteTopicPublisher(pTopPub, iError, pczError)
```

Description

Deletes the specified topic publisher.

Parameters

Name	Type	Description
pTopPub	SBYN_TopicPublisher*	Pointer to the TopicPublisher object.
iError	int*	Pointer to the error number.

Name	Type	Description
pczError	char*	Pointer to the error message text.

Return Value

None.

DeleteMessage

Syntax

```
DeleteMessage(pMsg, iError, pczError)
```

Description

Deletes the specified message.

Parameters

Name	Type	Description
pMsg	SBYN_Message*	Pointer to the message.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

None.

5.8.21. The WString Helper Interface

The **WString** structure (wide string—not part of the standard JMS API), is used to facilitate the handling of text strings.

The **WString** helper interface includes the following methods:

- [CharToWString](#) on page 452
- [DeleteWString](#) on page 453
- [WStringToChar](#) on page 453

Also see [“The WStringList Helper Interface” on page 454](#).

CharToWString

Syntax

```
CharToWString(pczStr, iError, pczError)
```

Description

Translates the specified character array to a **WString** (wide string) object.

Parameters

Name	Type	Description
pczStr	const char*	Pointer to the character array.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

SBYN_WString*

Pointer to the **WString** object containing the translation of the character array.

DeleteWString

Syntax

```
DeleteWString(WStr, iError, pczError)
```

Description

Deletes the specified **WString** (wide string) object.

Parameters

Name	Type	Description
WStr	SBYN_WString*	Pointer to the WString object.

Return Value

None.

WStringToChar

Syntax

```
WStringToChar(WStr, iError, pczError)
```

Description

Translates the specified **WString** (wide string) object to a character array.

Parameters

Name	Type	Description
WStr	SBYN_WString*	Pointer to the WString object.

Return Value

char*

Pointer to the character array holding the translation.

5.8.22. The WStringList Helper Interface

The **WStringList** structure (wide string list—not part of the standard JMS API), is used to facilitate the handling of text strings.

The **WStringList** helper interface includes the following methods:

- [DeleteWStringList](#) on page 454
- [GetPropertyName](#) on page 454

Also see [“The WString Helper Interface” on page 452.](#)

DeleteWStringList

Syntax

```
DeleteWString(WStrList)
```

Description

Deletes the specified **WStringList** object (list of wide strings).

Parameters

Name	Type	Description
WStrList	SBYN_WString*	Pointer to the WStringList object.

Return Value

None.

GetPropertyName

Syntax

```
GetPropertyName(msg, iError, pczError)
```

Description

Retrieves a list of property names defined for the specified message.

Parameters

Name	Type	Description
msg	SBYN_Message*	Pointer to the message.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

SBYN_WStringList*

Pointer to the **WStringList** object (a list of wide strings) holding the property names.

5.8.23. Error Codes and Messages in the C API for JMS

The C API for JMS defines error codes as shown in Table 16.

Table 16 Error Codes Defined in the C API for JMS

Error Code	Explanation
0x80040300 = 2147746560 decimal JE_CODE_E_GENERAL	JMS exception, unspecified.
0x80040301 = 2147746561 decimal JE_CODE_E_REALLOC	A JMS exception occurred as a result of memory reallocation.
0x80040302 = 2147746562 decimal JE_CODE_E_MALLOC	A JMS exception occurred as a result of memory allocation.
0x80040303 = 2147746563 decimal JE_CODE_E_CONNECTION	A JMS exception occurred in setting up a connection.
0x80040304 = 2147746564 decimal JE_CODE_E_CREATION	A JMS exception occurred while creating a JMS object.
0x80040305 = 2147746565 decimal JE_CODE_E_CLOSED_SOCKET	A JMS exception occurred because of a closed socket.
0x80040306 = 2147746566 decimal JE_CODE_E_EOF	Processing ended because the BytesMessage or StreamMessage ended unexpectedly.
0x80040307 = 2147746567 decimal JE_CODE_E_NOTREADABLE	Processing ended because the message could not be read.
0x80040308 = 2147746568 decimal JE_CODE_E_NOTWRITEABLE	Processing ended because the message could not be written.
0x80040309 = 2147746569 decimal JE_CODE_E_FORMAT	Processing ended because the JMS client attempted to use a data type not supported by the message (or a message property), or attempted to read message data (or a message property) as the wrong type.
0x8004030A = 2147746570 decimal JE_CODE_E_ROLLBACK	The attempt to commit the session was unsuccessful of a transaction being rolled back.
0x8004030B = 2147746571 decimal JE_CODE_E_STATE	Processing ended because a method was invoked at an illegal or inappropriate time or because the provider was not in an appropriate state for the requested operation.
0x8004030C = 2147746572 decimal JE_CODE_E_DESTINATION	Processing ended because the destination could not be understood or was found to be invalid.
0x8004030D = 2147746573 decimal JE_CODE_E_NOTIMPL	Processing ended because a feature or interface was not implemented.
0x8004030E = 2147746574 decimal JE_CODE_E_INDEX_OUT_OF_BOUNDS	Processing ended because an index of some sort (such as to an array, to a string, or to a vector) was found to be outside the valid range.
0x8004030F = 2147746575 decimal JE_CODE_E_NULL_POINTER	Processing ended because the pointer in a case where an object was required.
0x80040310 = 2147746576 decimal JE_CODE_E_INVLD_CLIENT_ID	Processing ended because the connection's client ID was rejected by the provider.

Table 16 Error Codes Defined in the C API for JMS (Continued)

Error Code	Explanation
0x80040311 = 2147746577 decimal JE_CODE_E_INVLD_SELECTOR	Processing ended because the message selector was found to be syntactically invalid.
0x80040312 = 2147746578 decimal JE_CODE_E_JMS_SECURITY	Processing was ended by JMS Security – for example, the provider rejected a name/password combination submitted by a client.
0x80040313 = 2147746579 decimal JE_CODE_E_RESOURCE_ALLOC	Processing ended because of the provider was unable to allocate resources required for the method/function.
0x80040314 = 2147746580 decimal JE_CODE_E_XA_IN_PROGRESS	Processing ended because a transaction was in progress.

5.8.24. Differences Between JMS Java API and JMS C API

The C API has a few differences compared to the Java API for JMS, as noted below.

Differences Between Java and C in the BytesMessage Interface

For methods **ReadBytes()** and **WriteBytes()**, Java defines one signature to read or write the entire array (making use of Java’s ability to easily determine the length of the given array), and another to read/write a portion of the array. Because C has no easy way to determine array length, the C API implements only the second functionality (reading or writing a portion of the array), and also provides a facility to see if more data exists.

Differences Between Java and C in the MapMessage Interface

The C API has not defined any equivalent for the Java **MapMessage** interface.

Differences Between Java and C in the MessageProducer Interface

The C API provides functions **GetJMS_ProducerID()** and **SetJMS_ProducerID**, which have no equivalent in the Java API.

Differences Between Java and C in Error Handling

Although C does not support exception-handling, the supplied wrappers for C and RPG use an error-handling scheme allows a JMS C++ exception to propagate to the application.

For each JMS C/RPG API, the final two parameters are an integer (*iErr*) and a text buffer (*szErrBuf*). Check the value of *iErr* after making any C API call; if the value is nonzero, it indicates that the underlying code has thrown an exception, and you should check the error description contained in the buffer.

For example:

```
pTopicConnection = CreateTopicConnection(pTcf, &iErr, szErrBuf);
```



```
if (iErr){  
    printf("ERROR %d: %s\n", iErr, szErrBuf);  
}
```

5.9 The Supported C++ APIs for Oracle Seebeyond JMS

The e*Gate API Kit contains C++ APIs that are used to extend the functionality of e*Gate. These APIs are contained in the following classes:

- [The Message Interface for JMS in C++](#) on page 458
- [The BytesMessage Interface for JMS in C++](#) on page 483
- [The TextMessage Class](#) on page 498
- [The Connection Interface for JMS in C++](#) on page 499
- [The QueueConnection Interface for JMS in C++](#) on page 501
- [The Session Interface for JMS in C++](#) on page 501
- [The TopicConnection Interface for JMS in C++](#) on page 504
- [The QueueConnectionFactory Interface for JMS in C++](#) on page 507
- [The TopicConnectionFactory Interface for JMS in C++](#) on page 507
- [The ExceptionListener Interface for JMS in C++](#) on page 508
- [The DeliveryMode Interface for JMS in C++](#) on page 509
- [The Queue Interface for JMS in C++](#) on page 509
- [The TemporaryQueue Interface for JMS in C++](#) on page 510
- [The Topic Interface for JMS in C++](#) on page 511
- [The TemporaryTopic Interface for JMS in C++](#) on page 512
- [The MessageProducer Interface for JMS in C++](#) on page 512
- [The QueueSender Interface for JMS in C++](#) on page 517
- [The TopicPublisher Interface](#) on page 521
- [The QueueSession Interface for JMS in C++](#) on page 525
- [The TopicSession Interface for JMS in C++](#) on page 527
- [The Xid Interface for JMS in C++](#) on page 532
- [The XAResource Interface for JMS in C++](#) on page 533

5.10 The C++ API for Oracle Seebeyond JMS

5.10.1. The Message Interface for JMS in C++

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

Most message-oriented middleware (MOM) products treat messages as lightweight entities that consist of a header and a payload. The header contains fields used for message routing and identification; the payload contains the application data being sent.

Within this general form, the definition of a message varies significantly across products. It would be quite difficult for the JMS API to support all of these message models.

With this in mind, the JMS message model has the following goals:

- Provide a single, unified message API
- Provide an API suitable for creating messages that match the format used by provider-native messaging applications
- Support the development of heterogeneous applications that span operating systems, machine architectures, and computer languages
- Support messages containing objects in the Java programming language ("Java objects")
- Support messages containing Extensible Markup Language (XML) pages

The `Message` methods are:

[acknowledge](#) on page 459

[clearProperties](#) on page 460

[getBooleanProperty](#) on page 460

[getDoubleProperty](#) on page 461

[getIntProperty](#) on page 462

[getPropertyName](#) on page 463

[getStringProperty](#) on page 463

[setByteProperty](#) on page 464

[setFloatProperty](#) on page 465

[setLongProperty](#) on page 466

[setShortProperty](#) on page 467

[propertyExists](#) on page 467

[getByteProperty](#) on page 468

[clearBody](#) on page 459

[propertyExists](#) on page 460

[getByteProperty](#) on page 461

[getFloatProperty](#) on page 461

[getLongProperty](#) on page 462

[getShortProperty](#) on page 463

[setBooleanProperty](#) on page 464

[setDoubleProperty](#) on page 465

[setIntProperty](#) on page 465

[setObjectProperty](#) on page 466

[setStringProperty](#) on page 467

[getBooleanProperty](#) on page 468

[getDoubleProperty](#) on page 469

- [getFloatProperty](#) on page 469
- [getLongProperty](#) on page 470
- [getShortProperty](#) on page 471
- [setBooleanProperty](#) on page 471
- [setDoubleProperty](#) on page 472
- [setIntProperty](#) on page 473
- [setObjectProperty](#) on page 474
- [setStringProperty](#) on page 475
- [getJMSCorrelationIDAsBytes](#) on page 475
- [getJMSExpiration](#) on page 476
- [getJMSPriority](#) on page 476
- [getJMSReplyTo](#) on page 477
- [getJMSType](#) on page 477
- [setJMSCorrelationIDAsBytes](#) on page 478

- [setJMSMessageID](#) on page 479
- [setJMSRedelivered](#) on page 480
- [setJMSTimestamp](#) on page 481
- [setJMSCorrelationIDAsBytes](#) on page 481
- [setJMSType](#) on page 482

- [getIntProperty](#) on page 469
- [getObjectProperty](#) on page 470
- [getStringProperty](#) on page 471
- [setByteProperty](#) on page 472
- [setFloatProperty](#) on page 473
- [setLongProperty](#) on page 473
- [setShortProperty](#) on page 474
- [getJMSCorrelationID](#) on page 475
- [getJMSDeliveryMode](#) on page 475
- [getJMSMessageID](#) on page 476
- [getJMSRedelivered](#) on page 477
- [getJMSTimestamp](#) on page 477
- [setJMSCorrelationID](#) on page 478
- [setJMSDeliveryMode](#) on page 478
- [setJMSExpiration](#) on page 479
- [setJMSPriority](#) on page 480
- [setJMSReplyTo](#) on page 480
- [setJMSType](#) on page 481
- [setJMSMessageID](#) on page 482

acknowledge

Syntax

```
void acknowledge()
```

Description

Acknowledges the receipt of current and previous messages.

Return Value

None.

clearBody

Syntax

```
void clearBody()
```

Description

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

Return Value

None.

clearProperties

Syntax

```
void clearProperties()
```

Description

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

Return Value

None.

propertyExists

Syntax

```
STCBOOL propertyExists(name)
```

Description

Checks whether a value for a specific property exists.

Parameters

Name	Type	Description
name	WString&	The name of the property to check.

Return Value

STCBOOL

Returns **true** if the property value is defined; otherwise, returns **false**.

getBooleanProperty

Syntax

```
STCBOOL getBooleanProperty(name)
```

Description

Checks whether a value for a specific property exists.

Parameters

Name	Type	Description
name	WString&	The name of the property to check.

Return Value

STCBOOL

Returns **true** if the property value is defined; otherwise, returns **false**.

getBytesProperty

Syntax

```
unsigned getBytesProperty(name)
```

Description

Reads the value of the specified byte property in the specified message.

Parameters

Name	Type	Description
named	WString&	The name of the property to check.

Return Value

unsigned char

The value of the property.

getDoubleProperty

Syntax

```
double getDoubleProperty(name)
```

Description

Reads the value of the specified double (numeric) property in the specified message.

Parameters

Name	Type	Description
name	WString&	The name of the property to check.

Return Value

Double

The value of the property.

getFloatProperty

Syntax

```
float getFloatProperty(name)
```

Description

Reads the value of the specified floating-point (numeric) property in the specified message.

Parameters

Name	Type	Description
name	WString&	The name of the property to check.

Return Value

float

The value of the property.

getIntProperty

Syntax

```
int getIntProperty(name)
```

Description

Reads the value of the specified integer (numeric) property in the specified message.

Parameters

Name	Type	Description
name	WString&	The name of the property to check.

Return Value

int

The value of the property.

getLongProperty

Syntax

```
int64_t getLongProperty(name)
```

Description

Reads the value of the specified long (numeric) property in the specified message.

Parameters

Name	Type	Description
name	WString&	The name of the property to check.

Return Value

int64_t

The long value of the property.

getPropertyName

Syntax

```
wNamesList getPropertyName (name)
```

Description

Returns a list of WStrings for the specified message.

Parameters

Name	Type	Description
name	wNamesList	Returns a list of the property names.

Return Value

wNamesList

Returns a list.

getShortProperty

Syntax

```
short getShortProperty (name)
```

Description

Reads the value of the specified short (numeric) property in the specified message.

Parameters

Name	Type	Description
name	WString&	The name of the property to check.

Return Value

short

The value of the property.

getStringProperty

Syntax

```
WString getStringProperty (name)
```

Description

Reads the value of the specified text property in the specified message.

Parameters

Name	Type	Description
name	WString	The name of the property to check.

Return Value

WString

A WString (wide string) object containing the value of the property.

setBooleanProperty

Syntax

```
void setBooleanProperty(name, value)
```

Description

Writes a value for the specified Boolean property in the specified message.

Parameters

Name	Type	Description
name	WString&	The name of the property to check.
value	STCBool	The value to be written.

Return Value

None

setByteProperty

Syntax

```
void setByteProperty(name, value)
```

Description

Writes a value for the specified byte property in the specified message.

Parameters

Name	Type	Description
name	WString&	The name of the property to check.
value	unsigned char	The value to be written.

Return Value

None

setDoubleProperty

Syntax

```
void setDoubleProperty(name, value)
```

Description

Writes a value for the specified double (numeric) property in the specified message.

Parameters

Name	Type	Description
name	WString&	The name of the property to check.
value	double	The value to be written.

Return Value

None

setFloatProperty

Syntax

```
void setFloatProperty(name, value)
```

Description

Writes a value for the specified byte property in the specified message.

Parameters

Name	Type	Description
name	WString&	The name of the property to check.
value	float	The value to be written.

Return Value

None

setIntProperty

Syntax

```
void setIntProperty(name, value)
```

Description

Writes a value for the specified byte property in the specified message.

Parameters

Name	Type	Description
name	WString&	The name of the property to check.
value	int	The value to be written.

Return Value

None

setLongProperty

Syntax

```
void setLongProperty(name, value)
```

Description

Writes a value for the specified long (numeric) property in the specified message.

Parameters

Name	Type	Description
name	WString&	The name of the property to check.
User defined	long	The value to be written.

Return Value

None

setObjectProperty

Syntax

```
void setObjectProperty(name, value)
```

Description

Writes a value for the specified object property in the specified message.

Parameters

Name	Type	Description
name	WString&	The name of the property to check.
User defined	SerialObject	The value to be written.

Return Value

None

setShortProperty

Syntax

```
void setShortProperty(name, value)
```

Description

Writes a value for the specified short (numeric) property in the specified message.

Parameters

Name	Type	Description
name	WString&	The name of the property to check.
value	short	The value to be written.

Return Value

None

setStringProperty

Syntax

```
void setStringProperty(name, value)
```

Description

Writes a value for the specified text property in the specified message.

Parameters

Name	Type	Description
name	WString&	The name of the property to check.
value	char *	The value to be written.

Return Value

None

propertyExists

Syntax

```
STCBOOL propertyExists(name)
```

Description

Checks whether a value for a specific property exists.

Parameters

Name	Type	Description
name	char*	Pointer to the name of the property to check.

Return Value

STCBOOL

Returns **true** if the property value is defined; otherwise, returns **false**.

getBooleanProperty

Syntax

```
STCBOOL getBooleanProperty(name)
```

Description

Reads the value of the specified Boolean property in the specified message.

Parameters

Name	Type	Description
name	char*	Pointer to the name of the property to check.

Return Value

STCBOOL

The value of the property.

getBytesProperty

Syntax

```
unsigned char getBytesProperty(name)
```

Description

Reads the value of the specified byte property in the specified message.

Parameters

Name	Type	Description
name	char*	Pointer to the name of the property to check.

Return Value

unsigned char

The value of the property.

getDoubleProperty

Syntax

```
double getDoubleProperty(name)
```

Description

Reads the value of the specified double (numeric) property in the specified message.

Parameters

Name	Type	Description
name	char*	Pointer to the name of the property to check.

Return Value

double

The value of the property.

getFloatProperty

Syntax

```
float getFloatProperty(name)
```

Description

Reads the value of the specified floating-point (numeric) property in the specified message.

Parameters

Name	Type	Description
name	char*	Pointer to the name of the property to check.

Return Value

float

The value of the property.

getIntProperty

Syntax

```
int getIntProperty(name)
```

Description

Reads the value of the specified integer (numeric) property in the specified message.

Parameters

Name	Type	Description
name	char*	Pointer to the name of the property to check.

Return Value

int

The value of the property.

getLongProperty

Syntax

```
int64_t getLongProperty(name)
```

Description

Reads the value of the specified long (numeric) property in the specified message.

Parameters

Name	Type	Description
name	char*	Pointer to the name of the property to check.

Return Value

int64_t

The long value of the property.

getObjectProperty

Syntax

```
SerialObject getObjectProperty(name)
```

Description

Reads the value of the specified object property in the specified message.

Parameters

Name	Type	Description
name	char*	Pointer to the name of the object property to check.

Return Value

SerialObject

The serialized value of the property.

getShortProperty

Syntax

```
short getShortProperty(name)
```

Description

Reads the value of the specified short (numeric) property in the specified message.

Parameters

Name	Type	Description
name	char*	Pointer to the name of the property to check.

Return Value

short

The value of the property.

getStringProperty

Syntax

```
WString getStringProperty(name)
```

Description

Reads the value of the specified text property in the specified message.

Parameters

Name	Type	Description
name	char*	Pointer to the name of the property to check.

Return Value

WString*

Pointer to a **WString** (wide string) object containing the value of the property.

setBooleanProperty

Syntax

```
void setBooleanProperty(name, value)
```

Description

Writes a value for the specified Boolean property in the specified message.

Parameters

Name	Type	Description
name	char*	Pointer to the property name.
value	STCBOOL	The value to be written.

Return Value

None.

setByteProperty

Syntax

```
void setByteProperty(name, value)
```

Description

Writes a value for the specified byte property in the specified message.

Parameters

Name	Type	Description
name	char*	Pointer to the property name.
value	unsigned char	The value to be written.

Return Value

None.

setDoubleProperty

Syntax

```
void setDoubleProperty(name, value)
```

Description

Writes a value for the specified double (numeric) property in the specified message.

Parameters

Name	Type	Description
name	char*	Pointer to the property name.
value	double	The value to be written.

Return Value

None.

setFloatProperty

Syntax

```
void setFloatProperty(name, value)
```

Description

Writes a value for the specified floating-point (numeric) property in the specified message.

Parameters

Name	Type	Description
name	char*	Pointer to the property name.
Value	float	The value to be written.

Return Value

None.

setIntProperty

Syntax

```
void setIntProperty(name, value)
```

Description

Writes a value for the specified integer (numeric) property in the specified message.

Parameters

Name	Type	Description
name	char*	Pointer to the property name.
value	int	The value to be written.

Return Value

None.

setLongProperty

Syntax

```
void setLongProperty(name, value)
```

Description

Writes a value for the specified long (numeric) property in the specified message.

Parameters

Name	Type	Description
name	char*	Pointer to the property name.
value	int64_t	The value to be written.

Return Value

None.

setObjectProperty

Syntax

```
void setObjectProperty(name, value)
```

Description

Writes a value for the specified long (numeric) property in the specified message.

Parameters

Name	Type	Description
name	char*	Pointer to the property name.
value	SerialObject	The value to be written.

Return Value

None.

setShortProperty

Syntax

```
void setShortProperty(name, value)
```

Description

Writes a value for the specified short (numeric) property in the specified message.

Parameters

Name	Type	Description
name	char*	Pointer to the property name.
value	short	The value to be written.

Return Value

None.

setStringProperty

Syntax

```
void setStringProperty(name, value)
```

Description

Writes a value for the specified text property in the specified message.

Parameters

Name	Type	Description
name	char*	Pointer to the property name.
value	char*	The value to be written.

Return Value

None.

getJMSCorrelationID

Syntax

```
WString getJMSCorrelationID()
```

Description

Retrieves the correlation ID for the specified message.

Return Value

WString*

Pointer to a **WString** (wide string) object containing the text.

getJMSCorrelationIDsAsBytes

Syntax

```
char* getJMSCorrelationIDsAsBytes()
```

Description

Retrieves the correlation ID for the specified message as an array of characters.

Return Value

None

getJMSDeliveryMode

Syntax

```
int getJMSDeliveryMode()
```

Description

Retrieves the value of the **DeliveryMode** property for the specified message.

Return Value

None

getJMSExpiration

Syntax

```
int64_t getJMSExpiration()
```

Description

Retrieves the value of the timestamp set for the expiration of the specified message.

Return Value

int64_t
Long timestamp.

getJMSMessageID

Syntax

```
WString getJMSMessageID()
```

Description

Retrieves the message ID of the specified message.

Return Value

WString*
Pointer to a **WString** (wide string) object containing the text.

getJMSPriority

Syntax

```
int getJMSPriority()
```

Description

Retrieves the priority level for the specified message.

Return Value

int
The priority level.

getJMSRedelivered

Syntax

```
STCBOOL getJMSRedelivered()
```

Description

Retrieves an indication of whether the specified message is being redelivered.

Return Value

STCBOOL

Returns **true** if the message is being redelivered; otherwise, returns **false**.

getJMSReplyTo

Syntax

```
Destination* getJMSReplyTo()
```

Description

Retrieves the **Destination** object where a reply to the specified message should be sent (for request/reply messaging).

Return Value

SBYN_Destination*

Pointer to the **Destination** object.

getJMSTimestamp

Syntax

```
int64_t getJMSTimestamp()
```

Description

Retrieves the timestamp of the specified message.

Return Value

int64_t

Long timestamp.

getJMSType

Syntax

```
WString getJMSType()
```

Description

Gets the message type identifier supplied by the client when the message was sent.

Return Value

WString*

Pointer to a **WString** (wide string) object containing the text.

setJMSCorrelationID

Syntax

```
void setJMSCorrelationID(correlationID)
```

Description

Sets the correlation ID for the specified message.

Parameters

Name	Type	Description
correlationID	WString	The text string containing the correlation ID.

Return Value

None.

setJMSCorrelationIDAsBytes

Syntax

```
void setJMSCorrelationIDAsBytes(deliveryMode)
```

Description

Sets the correlation ID for the specified message.

Parameters

Name	Type	Description
deliveryMode	char *	Pointer to the character array containing the bytes for the correlation ID.

Return Value

None.

setJMSDeliveryMode

Syntax

```
void setJMSDeliveryMode(destination)
```

Description

Sets the delivery mode for the specified message.

Parameters

Name	Type	Description
destination	Destination*	Pointer to the value corresponding to the delivery mode.

Return Value

None.

setJMSExpiration

Syntax

```
void setJMSExpiration(expiration)
```

Description

Sets the timestamp at which the specified message is due to expire.

Parameters

Name	Type	Description
expiration	int64_t	Timestamp of the expiration.

Return Value

None.

setJMSMessageID

Syntax

```
void setJMSMessageID(id)
```

Description

Sets the message ID of the specified message.

Parameters

Name	Type	Description
id	WString	The text string containing the message ID.

Return Value

None.

setJMSPriority

Syntax

```
void setJMSPriority(priority)
```

Description

Sets the priority level for the specified message.

Parameters

Name	Type	Description
priority	int	The priority level.

Return Value

None.

setJMSRedelivered

Syntax

```
void setJMSRedelivered(redelivered)
```

Description

Determines whether to flag the specified message as being redelivered. Used, for example, to specify redelivery for a message that has been sent but not acknowledged.

Parameters

Name	Type	Description
redelivered	STCBOOL	Flag: If true , the message is being redelivered.

Return Value

None.

setJMSReplyTo

Syntax

```
void setJMSReplyTo(replyTo)
```

Description

Sets the **Destination** object where a reply to the specified message should be sent.

Parameters

Name	Type	Description
replyto	Destination*	Pointer to the Destination object.

Return Value

None.

setJMSTimestamp

Syntax

```
void setJMSTimestamp(timestamp)
```

Description

Sets the timestamp (JMSTimestamp header field) that the specified message was handed off to a provider to be sent. Note that this is not necessarily the time the message is actually transmitted; the actual **send** can occur later because of transactions or other client-side queuing of messages.

Parameters

Name	Type	Description
timestamp	int64_t	Timestamp to be set.

Return Value

None.

setJMSType

Syntax

```
void setJMSType(type)
```

Description

Sets the JMSType header field, which is often used to identify the message structure and the payload type.

Parameters

Name	Type	Description
type	WString	The text string containing the data.

Return Value

None.

setJMSCorrelationIDAsBytes

Syntax

```
void setJMSCorrelationIDAsBytes(correlationID)
```

Description

Sets the correlation ID for the specified message.

Parameters

Name	Type	Description
correlationID	char*	Pointer to the character array containing the bytes for the correlation ID.

Return Value

None.

setJMSMessageID

Syntax

```
void setJMSMessageID(id)
```

Description

Sets the message ID of the specified message.

Parameters

Name	Type	Description
id	char*	Pointer to the text string containing the message ID.

Return Value

None.

setJMSType

Syntax

```
void setJMSType(type)
```

Description

Sets the JMSType header field, which is often used to identify the message structure and the payload type.

Parameters

Name	Type	Description
pczJMSType	char*	Pointer to the text string containing the data.

Return Value

None.

5.10.2. The `BytesMessage` Interface for JMS in C++

A `BytesMessage` object is used for messages containing a stream of uninterpreted bytes. The receiver of the message supplies the interpretation of the bytes.

The `BytesMessage` methods are:

[readBoolean](#) on page 483

[readDouble](#) on page 484

[readInt](#) on page 484

[readShort](#) on page 485

[readUnsignedShort](#) on page 485

[reset](#) on page 486

[writeByte](#) on page 487

[writeBytes](#) on page 487

[writeDouble](#) on page 488

[writeInt](#) on page 489

[writeShort](#) on page 490

[readByte](#) on page 483

[readChar](#) on page 484

[readFloat](#) on page 484

[readLong](#) on page 485

[readUnsignedByte](#) on page 485

[readUTF](#) on page 486

[writeBoolean](#) on page 486

[writeBytes](#) on page 487

[writeChar](#) on page 488

[writeFloat](#) on page 489

[writeLong](#) on page 489

`readBoolean`

Syntax

```
STCBOOL readBoolean()
```

Description

Reads a Boolean value from the `BytesMessage` stream.

Return Value

STCBOOL

The value read from the `BytesMessage` stream.

`readByte`

Syntax

```
unsigned char readByte()
```

Description

Reads a single unsigned character from the `BytesMessage` stream.

Return Value

unsigned char

The value read from the `BytesMessage` stream.

readChar

Syntax

```
unsigned short readChar()
```

Description

Reads a single Unicode character from the **BytesMessage** stream.

Return Value

unsigned short

The value read from the **BytesMessage** stream.

readDouble

Syntax

```
double readDouble()
```

Description

Reads a double numeric value from the **BytesMessage** stream.

Return Value

double

The value read from the **BytesMessage** stream.

readFloat

Syntax

```
float readFloat()
```

Description

Reads a floating-point numeric value from the **BytesMessage** stream.

Return Value

float

The value read from the **BytesMessage** stream.

readInt

Syntax

```
int readInt()
```

Description

Reads a signed integer value from the **BytesMessage** stream.

Return Value

int

The value read from the **BytesMessage** stream.

readLong

Syntax

```
int64_t readLong(pMsg, iError, pczError)
```

Description

Reads a signed long integer from the **BytesMessage** stream.

Return Value

long

The value read from the **BytesMessage** stream.

readShort

Syntax

```
short readShort(pMsg, iError, pczError)
```

Description

Reads a signed short integer from the **BytesMessage** stream.

Return Value

short

The value read from the **BytesMessage** stream.

readUnsignedByte

Syntax

```
readUnsignedByte()
```

Description

Reads an unsigned short integer from the **BytesMessage** stream.

Return Value

int

The value read from the **BytesMessage** stream.

readUnsignedShort

Syntax

```
int readUnsignedShort()
```

Description

Reads an unsigned short integer from the **BytesMessage** stream.

Return Value

int

The value read from the **BytesMessage** stream.

readUTF

Syntax

```
WString readUTF()
```

Description

Reads the value of a string that has been encoded using a modified UTF-8 format from the **BytesMessage** stream.

Return Value

WString*

Pointer to a **WString** (wide string) object containing the text from the **BytesMessage** stream.

reset

Syntax

```
void reset()
```

Description

Puts the message body into read-only mode and repositions the stream of bytes to the beginning.

Return Value

None.

writeBoolean

Syntax

```
void writeBoolean(value)
```

Description

Writes a Boolean value to the **BytesMessage** stream.

Parameters

Name	Type	Description
value	STCBOOL	The value to be written.

Return Value

None.

writeByte

Syntax

```
void writeByte(value)
```

Description

Writes a single byte (unsigned char) to the **BytesMessage** stream.

Parameters

Name	Type	Description
value	unsigned char	The value to be written.

Return Value

None.

writeBytes

Syntax

```
void writeBytes(value)
```

Description

Writes an array of bytes (unsigned char values) to the **BytesMessage** stream.

Parameters

Name	Type	Description
value	unsigned char*	Pointer to the value to be written.

Return Value

None.

writeBytes

Syntax

```
writeBytesEx(value, offset, length)
```

Description

Writes a portion of a byte array (unsigned char values) to the **BytesMessage** stream. For example, to extract "nag" from "manager", set `iOffset=2` and `iLength=3`.

Parameters

Name	Type	Description
pczValue	unsigned char*	Pointer to the value to be written.
offset	int	The initial offset within the byte array.
length	int	The number of bytes to use.

Return Value

None.

writeChar

Syntax

```
void writeChar(value)
```

Description

Writes an unsigned short integer to the **BytesMessage** stream.

Parameters

Name	Type	Description
value	unsigned short	The value to be written.

Return Value

None.

writeDouble

Syntax

```
void writeDouble(value)
```

Description

Writes a double numeric value to the **BytesMessage** stream.

Parameters

Name	Type	Description
value	double	The value to be written.

Return Value

None.

writeFloat

Syntax

```
writeFloat(value)
```

Description

Writes a floating-point numeric value to the **BytesMessage** stream.

Parameters

Name	Type	Description
value	float	The value to be written.

Return Value

None.

writeInt

Syntax

```
void writeInt(value)
```

Description

Writes an integer numeric value to the **BytesMessage** stream.

Parameters

Name	Type	Description
value	int	The value to be written.

Return Value

None.

writeLong

Syntax

```
void writeLong(value)
```

Description

Writes a long numeric value to the **BytesMessage** stream.

Parameters

Name	Type	Description
value	int	The value to be written.

Return Value

None.

writeShort

Syntax

```
void writeShort(value)
```

Description

Writes a short numeric value to the **BytesMessage** stream.

Parameters

Name	Type	Description
value	short	The value to be written.

Return Value

5.11 None. The MapMessage Class

A MapMessage is used to send a set of name-value pairs, where names are Strings. The entries are accessed sequentially or randomly by name. The order of the entries is undefined. It inherits from Message, and adds a map message body.

The methods of the MapMessage Class are:

getBoolean

Syntax

```
STCBOOL getBoolean(name)
```

Description

The getBoolean method returns the boolean value with the given name

Parameters

Name	Type	Description
name	WString.	The name of the boolean property.

Return Value

STCBOOL

getBytes

Syntax

```
unsigned char getBytes(name)
```

Description

The `getBytes` method returns the byte value with the given name.

Parameters

Name	Type	Description
<code>name</code>	WString	The name of the byte.

Return Value

unsigned char

getBytes

Syntax

```
unsigned char* getBytes(name)
```

Description

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

Parameters

Name	Type	Description
<code>name</code>	WString	The name of the byte byte array value.

Return Value

unsigned char*

Pointer to to the byte array value.

getBytes

Syntax

```
unsigned int getBytes(name, value, length)
```

Description

The `getBytes` method returns the int value with the given name as a variable.

Parameters

Name	Type	Description
name	WString	The name of the byte byte array value.
value	int	Value to be written.
length	int	The number of bytes to use.

Return Value

unsigned int

The value read.

getChar

Syntax

```
wchar_t getChar(name)
```

Description

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

Parameters

Name	Type	Description
name	WString	The name of the Unicode character.

Return Value

wchar_t

getDouble

Syntax

```
double getDouble(name)
```

Description

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

Parameters

Name	Type	Description
name	WString	The name of the double property.

Return Value

double

getFloat

Syntax

```
float getFloat(name)
```

Description

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

Parameters

Name	Type	Description
name	WString	The name of the float property.

Return Value

float

getInt

Syntax

```
int getInt(name)
```

Description

The getInt method returns the int value of the given name

Parameters

Name	Type	Description
name	WString	The value of the name .

Return Value

int

getLong

Syntax

```
int64_t getLong(name)
```

Description

The getLong method returns the long value of the given name.

Parameters

Name	Type	Description
name	WString	The long value of the name.

The GetObject Method

The GetProperty Method

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

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

Name	Description
name	Name of the currency property.

The GetShort Method

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

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

Name	Description
name	The name of the short currency property.

The GetString Method

Return the String value with the given name

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

Name	Description
name	The name of the String property.

The ItemExists Method

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

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

Name	Description
name	The name of the item to check.

The PropertyExists Method

The PropertyExists method checks if a property value exists.

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

Name	Description
name	The name of the property value.

The SetBoolean Method

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

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

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

The SetByte Method

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

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

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

The SetBytes Method

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

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

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

Name	Description
length	The number of bytes to use.

The SetChar Method

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

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

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

The SetDouble Method

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

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

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

The SetFloat Methods

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

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

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

The SetInt Method

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

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

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

The SetLong Method

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

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

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

The SetObject Method

This method is currently not supported.

The SetProperty Method

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

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

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

The SetShort Method

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

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

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

The SetString Method

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

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

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

5.11.1. The TextMessage Class

A TextMessage is used to send a message containing text. It adds a text message body. When a client receives a TextMessage, it is in read-only mode. If an attempt is made to write to the message while in read-only mode, an error is returned. However, if ClearBody is called first, then message can be then read from and written to.

The TextMessage functions include the following:

[GetText](#) on page 498

[SetText](#) on page 498

[SetText](#) on page 499

GetText

Syntax

```
WString getText()
```

Description

Retrieves the string containing the data associated with the message.

Return Value

WString
Wide string.

SetText

Syntax

```
void setText(buffer)
```

Description

Sets the string containing the data associated with the message.

Parameters

Name	Type	Description
Buffer	WString	The text string.

Return Value

None.

SetText

Syntax

```
void SetText(buffer)
```

Description

Sets the string containing the data associated with the message.

Parameters

Name	Type	Description
Buffer	char*	Pointer to the text string.

Return Value

None.

5.11.2. The Connection Interface for JMS in C++

A **Connection** object is an active connection to a JMS point-to-point provider or an active connection to a JMS pub/sub provider. A client uses a **Connection** to create one or more **Sessions** for producing and consuming messages.

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

[close](#) on page 499

[getClientID](#) on page 500

[setClientID](#) on page 500

[start](#) on page 500

[stop](#) on page 500

close

Syntax

```
void close()
```

Description

Closes the specified connection.

Return Value

None.

getClientID

Syntax

```
WString getClientID()
```

Description

Retrieves the client ID associated with the specified **Connection** object.

Return Value

WString*
WString (wide string) object containing the text.

setClientID

Syntax

```
void setClientID(ClientID)
```

Description

Sets the client ID to be associated with the specified **Connection** object. In other words, this method allows you to name the connection.

Parameters

Name	Type	Description
clientID	WString	Text string for the client ID.

Return Value

None.

start

Syntax

```
void start()
```

Description

Starts (or restarts) delivering incoming messages via the specified **Connection** object. If the connection is already started, the call is ignored without error.

Return Value

None.

stop

Syntax

```
void stop()
```

Description

Temporarily halts delivering incoming messages via the specified **Connection** object. If the connection is already stopped, the call is ignored without error.

Return Value

None.

5.11.3. The QueueConnection Interface for JMS in C++

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

The QueueConnection Interface methods:

[createQueueSession](#) on page 501

createQueueSession

Syntax

```
QueueSession* createQueueSession(transacted, acknowledgeMode)
```

Description

Creates a **Session** object.

Parameters

Name	Type	Description
transacted	SBYN_BOOL	Flag: If true , the session is transacted.
acknowledgeMode	int	Ignored if the session is transacted. If the session is not transacted, indicates whether the consumer or client acknowledges messages that it receives.

Return Value

SBYN_Session* pointer.

5.11.4. The Session Interface for JMS in C++

The Session Interface methods:

[close](#) on page 502

[commit](#) on page 502

[getTransacted](#) on page 502

[recover](#) on page 502

[rollback](#) on page 503

[bytesMessage](#) on page 503

[createTextMessage](#) on page 503

[createTextMessage](#) on page 504

close

Syntax

```
void close()
```

Description

Closes the specified session.

Note: Sessions should be closed when they are no longer needed.

Return Value

None.

commit

Syntax

```
void commit()
```

Description

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

Return Value

None.

getTransacted

Syntax

```
STCBOOL getTransacted()
```

Description

Queries whether the specified session is or is not transacted.

STCBOOL

Returns **true** if the session is transacted; otherwise, returns **false**.

recover

Syntax

```
void recover()
```

Description

Stops message delivery in the specified session, causes all messages that might have been delivered but not acknowledged to be marked as **redelivered**, and restarts message delivery with the oldest unacknowledged message. Note that redelivered messages need not be delivered in the exact order they were originally delivered.

Return Value

None.

rollback

Syntax

```
rollback(pSessn, iError, pczError)
```

Description

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

Return Value

None.

bytesMessage

Syntax

```
BytesMessage* bytesMessage()
```

Description

Creates a **BytesMessage**— an object used to send a message containing a stream of uninterpreted bytes.

Return Value

BytesMessage*
Pointer to the created message object.

createTextMessage

Syntax

```
TextMessage createTextMessage()
```

Description

Creates an uninitialized **TextMessage**— an object used to send a message containing a string to be supplied.

Return Value

TextMessage
The created message object.

createTextMessage

Syntax

```
TextMessage createTextMessage(stringBuffer)
```

Description

Creates an initialized **TextMessage**— an object used to send a message containing the supplied string.

Parameters

Name	Type	Description
stringBuffer	WString	stringBuffer to the Text message.

Return Value

TextMessage

5.11.5. The TopicConnection Interface for JMS in C++

The TopicConnection Interface methods are:

[createTopicSession](#) on page 504

[close](#) on page 505

[getClientID](#) on page 505

[setClientID](#) on page 506

[setClientID](#) on page 506

[getExceptionListener](#) on page 506

createTopicSession

Description

```
ConnectionConsumer createTopicSession(transacted, acknowledgeMode)
```

Syntax

Create a TopicSession

Parameters

Name	Description
transacted	If true, session is transacted.

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

Return Value

ConnectionConsumer

close

Syntax

```
void close()
```

Description

Closes the specified session.

Note: Sessions should be closed when they are no longer needed.

Return Value

None.

getClientID

Syntax

```
WString getClientID()
```

Description

Retrieves the client ID associated with the specified **Connection** object.

Return Value

WString*

Pointer to a **WString** (wide string) object containing the text.

setClientID

Syntax

```
void setClientID(clientID)
```

Description

Sets the client ID to be associated with the specified **Connection** object. In other words, this method allows you to name the connection.

Parameters

Name	Type	Description
clientID	void	The text string for the client ID.

Return Value

None.

setClientID

Syntax

```
void setClientID(clientID)
```

Description

Sets the client ID to be associated with the specified **Connection** object. In other words, this method allows you to name the connection.

Parameters

Name	Type	Description
clientID	char*	Pointer to the text string for the client ID.

Return Value

char *
Pointer to the text string to the client ID.

getExceptionListener

Syntax

```
ExceptionListener getExceptionListener()
```

Description

Gets the `ExceptionListener` object for this connection.

Return Value

ExceptionListener

5.11.6. The QueueConnectionFactory Interface for JMS in C++

Using point-to-point messaging, a client uses a **QueueConnectionFactory** object to create **QueueConnection** objects.

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

- [createQueueConnection](#) on page 507
- [createQueueConnection](#) on page 507

createQueueConnection

Syntax

```
QueueConnection* createQueueConnection()
```

Description

Constructs a **QueueConnection** object.

Return Value

QueueConnection*

Pointer to the **QueueConnection** object that was created.

createQueueConnection

Syntax

```
QueueConnection createQueueConnection(userName, password)
```

Description

Constructs a **Connection** object

Parameters

Name	Type	Description
userName	char*	Pointer to the userName.
password	char*	Pointer to the password.

Return Value

QueueConnection* pointer.

5.11.7. The TopicConnectionFactory Interface for JMS in C++

Using pub/sub messaging, a client uses a **TopicConnectionFactory** object to create **TopicConnection** objects.

The **TopicConnectionFactory** interface includes the following methods:

- [createTopicConnectionFactory](#) on page 508

createTopicConnectionFactory

Syntax

```
TopicConnectionFactory* createTopicConnectionFactory()
```

Description

Constructs a **TopicConnectionFactory** for the specified host and port. Once constructed, it can create **TopicConnection** objects for a pub/sub JMS provider.

Return Value

TopicConnectionFactory*

Pointer to the **TopicConnection** object that was created.

5.11.8. The ExceptionListener Interface for JMS in C++

If the JMS IQ manager detects a serious problem with a Connection object, it informs the Connection object's ExceptionListener, if one has been registered. It does this by calling the listener's onException method, passing it a JMSEException argument describing the problem.

This allows a client to be asynchronously notified of a problem. Some Connections only consume messages so they would have no other way to learn their Connection has failed.

A JMS provider should attempt to resolve connection problems themselves prior to notifying the client of them.

The ExceptionListener Interface methods:

[OnException](#) on page 508

OnException

Syntax

```
void OnException(exception)
```

Description

Notifies user of a JMS exception.

Parameters

Name	Description
exception	The JMS exception.

Returns

None.

5.11.9. The DeliveryMode Interface for JMS in C++

The delivery modes supported by the JMS API are PERSISTENT and NON_PERSISTENT.

A client marks a message as persistent if it feels that the application will have problems if the message is lost in transit. A client marks a message as non-persistent if an occasional lost message is tolerable. Clients use delivery mode to tell the JMS IQ manager how to balance message transport reliability throughput.

Delivery mode only covers the transport of the message to its destination. Retention of a message at the destination until its receipt is acknowledged is not guaranteed by a PERSISTENT delivery mode. Clients should assume that message retention policies are set administratively. Message retention policy governs the reliability of message delivery from destination to message consumer. For example, if a client's message storage space is exhausted, some messages as defined by a site specific message retention policy may be dropped.

A message is guaranteed to be delivered once-and-only-once by a JMS Provider if the delivery mode of the message is persistent and if the destination has a sufficient message retention policy.

NON_PERSISTENT Field

This is the lowest overhead delivery mode because it does not require that the message be logged to stable storage. The level of JMS provider failure that causes a NON_PERSISTENT message to be lost is not defined.

A JMS provider must deliver a NON_PERSISTENT message with an at-most-once guarantee. This means it may lose the message but it must not deliver it twice.

```
public static final int NON_PERSISTENT
```

PERSISTENT Field

This mode instructs the JMS provider to log the message to stable storage as part of the client's send operation. Only a hard media failure should cause a PERSISTENT message to be lost.

5.11.10. The Queue Interface for JMS in C++

A Queue object encapsulates a provider-specific queue name. In this manner, a client specifies the identity of queue to JMS methods. The actual length of time messages are held by a queue and the consequences of resource overflow are not defined by JMS.

The Queue Interface methods are:

[getQueueName](#) on page 510

[toString](#) on page 510

getQueueName

Syntax

```
WString getQueueName()
```

Description

Get the name of this queue. Clients that depend upon the name, are not portable.

Returns

WString
Wide string object.

toString

Syntax

```
WString toString()
```

Description

Return a pretty printed version of the queue name

Returns

WString
Wide string object.

5.11.11. The TemporaryQueue Interface for JMS in C++

A TemporaryQueue is a unique Queue object created for the duration of a QueueConnection. It is a system defined queue that can only be consumed by the QueueConnection that created it.

The TemporaryQueue Interface methods are:

[Delete](#) on page 510

Delete

Syntax

```
void Delete()
```

Description

Delete this temporary queue. If there are still existing senders or receivers still using it, then a JMSEException will be thrown.

Throws `JMSEException` if JMS implementation fails to delete a Temporary topic due to some internal error.

Returns

Nothing.

5.11.12. The Topic Interface for JMS in C++

A Topic object encapsulates a provider-specific topic name. The topic object provides the means for a client to specify the identity of a topic to JMS methods.

Many Pub/Sub implementations group topics into hierarchies and provide various options for subscribing to parts of the hierarchy. JMS places no restriction on what a Topic object represents.

The Topic Interface methods:

[getTopicName](#) on page 511

[toString](#) on page 511

getTopicName

Syntax

```
WString getTopicName ()
```

Description

Gets the name of this topic.

Returns

WString
Wide string object.

toString

Syntax

```
WString toString()
```

Description

Returns a string representation of this object.

Returns

WString
Wide string object.

5.11.13. The TemporaryTopic Interface for JMS in C++

A TemporaryTopic object is a unique Topic object created for the duration of a TopicConnection. It is a system-defined topic that can be consumed only by the TopicConnection that created it.

The TemporaryTopic Interface methods are:

[Delete](#) on page 512

Delete

Syntax

```
void Delete()
```

Description

Deletes this temporary topic. If there are existing subscribers still using it, a `JMSException` will be thrown.

Returns

None.

5.11.14. The MessageProducer Interface for JMS in C++

The `MessageProducer` interface includes the following methods:

- [close](#) on page 513
- [getDeliveryMode](#) on page 513
- [getDisableMessageID](#) on page 513
- [getDisableMessageTimestamp](#) on page 513
- [getJMS_ProducerID](#) on page 514
- [getPriority](#) on page 514
- [getTimeToLive](#) on page 514
- [setDeliveryMode](#) on page 514
- [setDisableMessageID](#) on page 515
- [setDisableMessageTimestamp](#) on page 515
- [setJMS_ProducerID](#) on page 516
- [setPriority](#) on page 516
- [setTimeToLive](#) on page 516

close

Syntax

```
void close()
```

Description

Closes the specified message producer.

Note: When a message producer is no longer needed, it should be closed.

Return Value

None.

getDeliveryMode

Syntax

```
int QueueSenderGetDeliveryMode()
```

Description

Retrieves the value of the **DeliveryMode** property of the specified message producer.

Return Value

int

getDisableMessageID

Syntax

```
STCBOOL getDisableMessageID()
```

Description

Queries whether message IDs are or are not disabled for the specified message producer.

Return Value

SBYN_BOOL

Returns **true** if message IDs are disabled; otherwise, returns **false**.

getDisableMessageTimestamp

Syntax

```
STCBOOL getDisableMessageTimestamp()
```

Description

Queries whether message timestamping is or is not disabled for the specified message producer.

Return Value

SBYN_BOOL

Returns **true** if message timestamping is disabled; otherwise, returns **false**.

getJMS_ProducerID

Syntax

```
void getJMS_ProducerID(ProducerID)
```

Description

Retrieves the value of the **ProducerID** property for the specified message producer.

Return Value

None.

getPriority

Syntax

```
int getPriority()
```

Description

Queries the value of the message **Priority** property of the specified message producer.

Return Value

int
Message priority level, from **0** (least expedited) through **9** (most expedited).

getTimeToLive

Syntax

```
int64_t getTimeToLive()
```

Description

Queries the value of the **TimeToLive** property of the specified message producer.

Return Value

int64_t
Length of time (milliseconds elapsed since dispatch time) that a produced message should be retained by JMS.

setDeliveryMode

Syntax

```
void setDeliveryMode(DeliveryMode)
```

Description

Sets the value of the **DeliveryMode** property of the specified message producer.

Parameters

Name	Type	Description
DeliveryMode	int	Value for the DeliveryMode property.

Return Value

None.

setDisableMessageID

Syntax

```
void setDisableMessageID(value)
```

Description

Determines whether message IDs are disabled for this queue sender. Default **false**.

Parameters

Name	Type	Description
value	STC_BOOL	Flag, default false ; if true , message IDs are disabled.

Return Value

None.

setDisableMessageTimestamp

Syntax

```
void setDisableMessageTimestamp(value)
```

Description

Determines whether message timestamping is disabled for this message producer. Default **false**.

Since message timestamps take effort to create and increase the size of a message, this flag can be set **true** to reduce overhead if message IDs are not used by an application.

Parameters

Name	Type	Description
value	STC_BOOL	Flag, default false ; if true , message timestamping is disabled.

Return Value

None.

setJMS_ProducerID

Syntax

```
void setJMS_ProducerID(ProducerID)
```

Description

Sets the value of the **ProducerID** property for the specified message producer.

Parameters

Name	Type	Description
ProducerID	char*	Pointer to text string containing the Producer ID.

Return Value

None.

setPriority

Syntax

```
void setPriority(deliveryMode)
```

Description

Sets the value of the message **Priority** property, from **0** (least expedited) through **9** (most expedited).

Parameters

Name	Type	Description
deliverMode	int	Message priority level.

Return Value

None.

setTimeToLive

Syntax

```
void setTimeToLive(TimeToLive)
```

Description

Sets the value of the **TimeToLive** property of the specified message producer.

Parameters

Name	Type	Description
TimeToLive	long	Value to be used for TimeToLive: Length of time (milliseconds elapsed since dispatch time) that a produced message should be retained by JMS.

Return Value

None.

5.11.15. The QueueSender Interface for JMS in C++

Using point-to-point messaging, a client uses a **QueueSender** object to send messages to a queue. After sending a message to a queue, a client may retain the message and modify it without affecting the message that has been sent. The same message object may be sent multiple times.

The **QueueSender** interface includes the following methods:

- [send](#) on page 517
- [send](#) on page 518
- [send](#) on page 518
- [send](#) on page 518
- [send](#) on page 519
- [send](#) on page 519
- [send](#) on page 520
- [send](#) on page 520

send

Syntax

```
void send(message)
```

Description

Sends the specified message to the queue.

Parameters

Name	Type	Description
message	Message*	Pointer to the message to send.

Return Value

None.

send

Syntax

```
void send(message)
```

Description

Sends the specified message to the queue.

Parameters

Name	Type	Description
message	BytesMessage*	Pointer to the message to send.

Return Value

None.

send

Syntax

```
void send(message)
```

Description

Sends the specified message to the queue.

Parameters

Name	Type	Description
message	TextMessage*	Pointer to the message to send.

Return Value

None.

send

Syntax

```
void send(message, DeliveryMode, priority, timeToLive)
```

Description

Sends the specified message to the queue, overriding one or more default values for properties of the specified queue sender.

Parameters

Name	Type	Description
message	message*	Pointer to the message to send.

Name	Type	Description
deliveryMode	int	Value to be used for DeliveryMode .
priority	int	Value to be used for Priority .
timeToLive	int64_t	Value to be used for TimeToLive : Length of time (milliseconds elapsed since dispatch time) that a produced message should be retained by JMS.

Return Value

None.

send

Syntax

```
void send(message, DeliveryMode, priority, timeToLive)
```

Description

Sends the specified message to the queue, overriding one or more default values for properties of the specified queue sender.

Parameters

Name	Type	Description
message	BytesMessage*	Pointer to the message to send.
deliveryMode	int	Value to be used for DeliveryMode .
priority	int	Value to be used for Priority .
timeToLive	int64_t	Value to be used for TimeToLive : Length of time (milliseconds elapsed since dispatch time) that a produced message should be retained by JMS.

Return Value

None.

send

Syntax

```
void send(message, DeliveryMode, priority, timeToLive)
```

Description

Sends the specified message to the queue, overriding one or more default values for properties of the specified queue sender.

Parameters

Name	Type	Description
message	TextMessage*	Pointer to the message to send.
deliveryMode	int	Value to be used for DeliveryMode .
priority	int	Value to be used for Priority .
timeToLive	int64_t	Value to be used for TimeToLive : Length of time (milliseconds elapsed since dispatch time) that a produced message should be retained by JMS.

Return Value

None.

send

Syntax

```
void send(queue, message)
```

Description

Sends the specified message to the specified queue, using default values for properties (**DeliveryMode**, **Priority**, and **TimeToLive**) of the specified queue sender.

Typically, a message producer is assigned a queue at creation time; however, the JMS API also supports unidentified message producers, which require that the queue be supplied every time a message is sent.

Parameters

Name	Type	Description
queue	Queue*	Pointer to the QueueSender object.
message	Message*	Pointer to the message to send.

Return Value

None.

send

Syntax

```
send(queue, message, deliveryMode, priority, timeToLive)
```

Description

Sends the specified message to the specified queue, overriding one or more default values for properties of the specified queue sender.

Parameters

Name	Type	Description
queue	Queue*	Pointer to the QueueSender object.
message	Message*	Pointer to the message to send.
deliveryMode	int	Value to be used for DeliveryMode .
priority	int	Value to be used for Priority .
timeToLive	int64_t	Value to be used for TimeToLive : Length of time (milliseconds elapsed since dispatch time) that a produced message should be retained by JMS.

Return Value

None.

5.11.16. The TopicPublisher Interface

The **TopicPublisher** interface includes the following methods:

- [getTopic](#) on page 521
- [publish](#) on page 522
- [publish](#) on page 522
- [publish](#) on page 522
- [publish](#) on page 523
- [publish](#) on page 523
- [publish](#) on page 524
- [publish](#) on page 524
- [publish](#) on page 525

getTopic

Syntax

```
Topic* getTopic()
```

Description

Gets the specified topic.

Returns

Topic*
Pointer to the specified topic.

publish

Syntax

```
void publish(message)
```

Description

Publishes the specified message to the topic, using default values for properties (**DeliveryMode**, **Priority**, and **TimeToLive**) of the specified topic publisher.

Parameters

Name	Type	Description
message	Message*	Pointer to the message to be published.

Return Value

None.

publish

Syntax

```
void publish(message)
```

Description

Publishes the specified message to the topic, using default values for properties (**DeliveryMode**, **Priority**, and **TimeToLive**) of the specified topic publisher.

Parameters

Name	Type	Description
message	BytesMessage*	Pointer to the message to be published.

Return Value

None.

publish

Syntax

```
void publish(message)
```

Description

Publishes the specified message to the topic, using default values for properties (**DeliveryMode**, **Priority**, and **TimeToLive**) of the specified topic publisher.

Parameters

Name	Type	Description
message	TextMessage*	Pointer to the message to be published.

Return Value

None.

publish

Syntax

```
void publish(message, deliveryMode, priority, timeToLive)
```

Description

Publishes the specified message to the topic, overriding one or more default values for properties of the specified topic publisher.

Parameters

Name	Type	Description
message	Message*	Pointer to the message to be published.
deliveryMode	int	Value to be used for DeliveryMode .
priority	int	Value to be used for Priority .
timeToLive	int64_t	Value to be used for TimeToLive : Length of time (milliseconds elapsed since dispatch time) that a produced message should be retained by JMS.

Return Value

None.

publish

Syntax

```
void publish(message, deliveryMode, priority, timeToLive)
```

Description

Publishes the specified message to the topic, overriding one or more default values for properties of the specified topic publisher.

Parameters

Name	Type	Description
message	BytesMessage*	Pointer to the message to be published.
deliveryMode	int	Value to be used for DeliveryMode .

Name	Type	Description
priority	int	Value to be used for Priority .
timeToLive	int64_t	Value to be used for TimeToLive : Length of time (milliseconds elapsed since dispatch time) that a produced message should be retained by JMS.

Return Value

None.

publish

Syntax

```
void publish(message, deliveryMode, priority, timeToLive)
```

Description

Publishes the specified message to the topic, overriding one or more default values for properties of the specified topic publisher.

Parameters

Name	Type	Description
message	TextMessage*	Pointer to the message to be published.
deliveryMode	int	Value to be used for DeliveryMode .
priority	int	Value to be used for Priority .
timeToLive	int64_t	Value to be used for TimeToLive : Length of time (milliseconds elapsed since dispatch time) that a produced message should be retained by JMS.

Return Value

None.

publish

Syntax

```
void publish(topic, message)
```

Description

Publishes the specified message to the specified topic.

Parameters

Name	Type	Description
topic	Topic*	Pointer to the TopicPublisher object.
message	Message*	Pointer to the message.

Return Value

None.

publish

Syntax

```
void publish(topic, message, deliveryMode, priority, timeToLive)
```

Description

Publishes the specified message to the specified topic, overriding one or more default values for properties of the specified topic publisher.

Parameters

Name	Type	Description
topic	Topic*	Pointer to the TopicPublisher object.
message	Message*	Pointer to the message to publish.
delivery	int	Value to be used for DeliveryMode .
priority	int	Value to be used for Priority .
timeToLive	long	Value to be used for TimeToLive : Length of time (milliseconds elapsed since dispatch time) that a produced message should be retained by JMS.

Return Value

None.

5.11.17. The QueueSession Interface for JMS in C++

The QueueSession Interface methods are:

[createQueue](#) on page 525

[createReceiver](#) on page 526

[createReceiver](#) on page 526

[createSender](#) on page 527

[createTemporaryQueue](#) on page 527

createQueue

Syntax

```
queue createQueue(queueName)
```

Description

Creates an identity with a specific queue name; does not create a physical queue.

This functionality is provided for rare cases where clients need to dynamically create a queue identity with a provider-specific name. Clients that depend on this functionality are not portable.

Parameters

Name	Type	Description
queueName	WString	The name of the queue.

Return Value

WString.

createReceiver

Syntax

```
QueueReceiver* createReceiver(queue)
```

Description

Creates a **Receiver** object to receive messages;

Parameters

Name	Type	Description
queue	queue*	Pointer to the queue.

Return Value

QueueReceiver* pointer.

createReceiver

Syntax

```
QueueReceiver* createReceive(queue, messageSelector)
```

Description

Creates a **Receiver** object to receive messages using a message selector.

Parameters

Name	Type	Description
queue	Queue*	Pointer to the queue.
messageSelector	char*	Pointer to the text of the message selector.

Return Value

QueueReceiver* pointer.

createSender

Syntax

```
QueueSender createSender(queue)
```

Description

Creates a **Sender** object to send messages.

Parameters

Name	Type	Description
queue	queue*	Pointer to the queue.

Return Value

QueueSender* pointer.

createTemporaryQueue

Syntax

```
TemporaryQueue createTemporaryQueue()
```

Description

Creates a **Temporary** object for a specified session.

Return Value

TemporaryQueue.

5.11.18. The TopicSession Interface for JMS in C++

The TopicSession Interface methods are:

[createDurableSubscriber](#) on page 528

[createDurableSubscriber](#) on page 528

[createPublisher](#) on page 529

[createSubscriber](#) on page 529

[createSubscriber](#) on page 530

[createTemporaryTopic](#) on page 530

[createTopic](#) on page 531

[unsubscribe](#) on page 531

createDurableSubscriber

Syntax

```
TopicSubscriber* createDurableSubscriber(topic, name)
```

Description

Creates a durable subscriber to the specified topic, specifying whether messages published by its own connection should be delivered to it.

Using pub/sub messaging, if a client needs to receive all the messages published on a topic, including messages published while the subscriber is inactive, it uses a *durable subscriber*. The JMS provider retains a record of this durable subscription and ensures that each message from the topic's publishers is retained until either it has been acknowledged by this durable subscriber or else it has expired.

Sessions with durable subscribers must always provide the same client ID, and each client must specify a name that (within the given client ID) uniquely identifies each durable subscription it creates. Only one session at a time can have a TopicSubscriber for a particular durable subscription. An *inactive* durable subscriber is one that exists but does not currently have a message consumer associated with it.

A client can change an existing durable subscription by creating a durable TopicSubscriber with the same name and a new topic (and/or *message selector*). Changing a durable subscriber is equivalent to deleting the old one and creating a new one.

Parameters

Name	Type	Description
topic	Queue*	Pointer to the topic.
name	char*	Pointer to the text string containing the client ID of the durable subscriber.

Return Value

TopicSubscriber*

Pointer to the created **TopicSubscriber** object.

createDurableSubscriber

Syntax

```
TopicSubscriber* createDurableSubscriber(topic, name,  
messageSelector, noLocal)
```

Description

Creates a durable subscriber to the specified topic, using a message selector (*messageSelector*) and/or specifying whether messages published by its own connection should be delivered to it (*noLocal*).

Parameters

Name	Type	Description
topic	Topic*	Pointer to the topic .
name	char*	Pointer to the text string containing the client ID of the durable subscriber.
messageSelector	char*	Pointer to the string containing the text of the message selector; only messages with properties matching the message selector expression are delivered. A value of null or an empty string indicates that there is no message selector for the message consumer.
noLocal	STCBOOL	Flag, default false . If true , inhibits the delivery of messages published by its own connection.

Return Value

TopicSubscriber*

Pointer to the created **TopicSubscriber** object.

createPublisher

Syntax

```
TopicPublisher createPublisher(topic)
```

Description

Creates a publisher for the specified topic.

Parameters

Name	Type	Description
topic	Topic*	Pointer to the topic .

Return Value

TopicPublisher*

Pointer to the created **TopicPublisher** object.

createSubscriber

Syntax

```
TopicSubscriber* createSubscriber(topic)
```

Description

Creates a *nondurable* subscriber to the specified topic—in other words, a subscriber that receives only those messages that are published while the subscriber is active.

Parameters

Name	Type	Description
topic	Topic*	Pointer to the topic .

Return Value

TopicSubscriber*

Pointer to the **TopicSubscriber** object.

createSubscriber

Syntax

```
TopicSubscriber* createSubscriber(topic, messageSelector, noLocal)
```

Description

Creates a *nondurable* subscriber to the specified topic—in other words, a subscriber that receives only those messages that are published while the subscriber is active

In some cases, a connection may both publish and subscribe to a topic. The `NoLocal` parameter allows a subscriber to inhibit the delivery of messages published by its own connection. The default value for this attribute is **false**.

Parameters

Name	Type	Description
topic	Topic*	Pointer to the topic .
messageSelector	char*	Pointer to the string containing the text of the message selector; only messages with properties matching the message selector expression are delivered. A value of null or an empty string indicates that there is no message selector for the message consumer.
noLocal	STCBOOL	Flag, default false . If true , inhibits the delivery of messages published by its own connection.

Return Value

TopicSubscriber*

Pointer to the **TopicSubscriber** object.

createTemporaryTopic

Syntax

```
TemporaryTopic* createTemporaryTopic()
```

Description

Creates a temporary topic that lives only as long as the specified TopicSession does (unless the topic is deleted earlier).

Return Value

TemporaryTopic*

Pointer to the created **TemporaryTopic** object.

createTopic

Syntax

```
Topic createTopic(topicName)
```

Description

Creates a topic identity with a specific topic name; does *not* create a physical topic.

This functionality is provided for rare cases where clients need to dynamically create a topic identity with a provider-specific name. Clients that depend on this functionality are not portable.

Parameters

Name	Type	Description
topicName	WString	The text string containing the name of the topic.

Return Value

Topic

unsubscribe

Syntax

```
void unsubscribe(name)
```

Description

Unsubscribes a durable subscription that has been created by a client. Note that it is an error to delete a durable subscription while there is an active TopicSession for the subscription, or while a consumed message is part of a pending transaction or has not been acknowledged in the session.

Parameters

Name	Type	Description
name	void	The name used to identify this subscription.

Return Value

None.

5.11.19. The Xid Interface for JMS in C++

The Xid Interface has the following methods:

[getBranchQualifier](#) on page 532

[getFormatId](#) on page 532

[getGlobalTransactionId](#) on page 532

getBranchQualifier

Syntax

```
const unsigned char* getBranchQualifier(&pl)
```

Description

Obtain the transaction branch identifier part of XID as an array of bytes.

Parameters

Name	Type	Description
&pl	long	Returns the long address.

Return Value

```
const unsigned char*
```

getFormatId

Syntax

```
uint64_t getFormatId()
```

Description

Obtain the format identifier part of the XID.

Return Value

```
uint64_t
```

getGlobalTransactionId

Syntax

```
const unsigned char* getGlobalTransactionId(&pl)
```

Description

Obtain the global transaction identifier part of XID as an array of bytes.

Parameters

Name	Type	Description
&pl	long	Returns the long address.

Return Value

const unsigned char*

5.11.20. The XAResource Interface for JMS in C++

The XAResource Interface has the following methods:

[commit](#) on page 533

[Xid**recover](#) on page 534

[rollback](#) on page 534

[getTransactionTimeout](#) on page 535

[setTransactionTimeout](#) on page 535

[isSameRM](#) on page 535

[prepare](#) on page 536

[start](#) on page 536

[end](#) on page 537

Flag definitions

TMENDRSCAN	End a recovery scan
TMFAIL	Disassociates the caller and mark the transaction branch rollback-only.
TMJOIN	Caller is joining existing transaction branch.
TMNOFLAGS	Use TMNOFLAGS to indicate no flags value is selected.
TMONEPHASE	Caller is using one-phase optimization.
TMRESUME	Caller is resuming association with suspended transaction branch.
TMSTARTRSCAN	Start a recovery scan.
TMSUCCESS	Disassociate caller from transaction branch
TMSUSPEND	Caller is suspending (not ending) association with transaction branch.
XA_OK	The transaction work has been prepared normally.
XA_RDONLY	The transaction branch has been read-only and has been committed.

commit

Syntax

```
void commit(Xid, onePhase)
```

Description

Commits the global transaction specified by `xid`

Parameters

Name	Type	Description
<code>xid</code>	<code>Xid*</code>	A pointer to the global transaction identifier.
<code>onePhase</code>	<code>STCBOOL</code>	Flag; if true, a one-phase commit protocol is used to commit the work done on behalf of <code>xid</code> .

Returns

`void`.

Xid**recover

Syntax

```
void Xid**recover(flag)
```

Description

This method is used during recovery to obtain the list of transaction branches that are currently in prepared or heuristically completed states.

Parameters

Name	Type	Description
<code>flag</code>	<code>int</code>	One of <code>TMSTARTRSCAN</code> , <code>TMENDRSCAN</code> , <code>TMNOFLAGS</code> . <code>TMNOFLAGS</code> must be used when no other flags are set in flags.

Returns

`int`.

rollback

Syntax

```
void rollback(xid)
```

Description

Roll back work done on behalf of a transaction branch.

Parameters

Name	Type	Description
xid	Xid*	A global transaction identifier

Returns

None.

getTransactionTimeout

Syntax

```
int getTransactionTimeout()
```

Description

Obtain the current transaction timeout value set for this `XAResource` instance. If `XAResource.setTransactionTimeout` was not used prior to calling this method, the return value is the default timeout set for the resource manager; otherwise, the value used in the previous `setTransactionTimeout` call is returned.

Returns

None.

setTransactionTimeout

Syntax

```
STCBOOL setTransactionTimeout()
```

Description

Sets the current transaction timeout value for this `XAResource` instance. Once set, this timeout value is effective until `setTransactionTimeout` is invoked again with a different value. To reset the timeout value to the default value used by the resource manager, set the value to zero. If the timeout operation is performed successfully, the method returns *true*; otherwise *false*. If a resource manager does not support transaction timeout value to be set explicitly, this method returns *false*.

Returns

STCBOOL

isSameRM

Syntax

```
int isSameRM(xares)
```

Description

This method is called to determine if the resource manager instance represented by the target object is the same as the resource manager instance represented by the parameter *xares*.

Parameters

Name	Type	Description
xid	Xid*	A global transaction identifier

Returns

int

prepare

Syntax

```
Xid* prepare(xid)
```

Description

Asks the resource manager to prepare for a transaction commit of the transaction specified in xid.

Parameters

Name	Type	Description
xid	Xid*	A global transaction identifier

Returns

int

A value indicating the resource manager's decision on the outcome of the transaction. The possible values are XA_RDONLY or XA_OK.

start

Syntax

```
void start(xid, flags)
```

Description

Start work on behalf of a transaction branch specified in xid. If TMJOIN is specified, the start is for joining a transaction previously seen by the resource manager. If TMRESUME is specified, the start is to resume a suspended transaction specified in the parameter xid. If neither TMJOIN or TMRESUME is specified and the transaction specified by xid has previously been seen by the JMS, the resource manager throws the XAException exception.

Parameters

Name	Type	Description
xid	Xid*	A global transaction identifier
flags	int	One of TMNOFLAGS, TMJOIN, or TMRESUME

Returns

None

end

Syntax

```
void end(xid, flags)
```

Description

Ends the work performed on behalf of a transaction branch. The JMS IQ manager disassociates the XA resource from the transaction branch specified and allows the transaction be completed.

If TMSUSPEND is specified in flags, the transaction branch is temporarily suspended in an incomplete state. The transaction context must then be resumed by specifying TMRESUME.

If TMFAIL is specified, the message failed. The JMS IQ manager may mark the transaction as rollback-only.

If TMSUCCESS is specified, the message has completed successfully.

Parameters

Name	Type	Description
xid	Xid*	A global transaction identifier
flags	int	One of TMSUCCESS, TMFAIL, or TMSUSPEND

Returns

None.

5.11.21. MSGSRVC_API *Lookup

The methods for the MSGSRVC_API *Lookup are:

[*LookupQueueConnectionFactory](#) on page 538

[*LookupXAQueueConnectionFactory](#) on page 538

[*LookupQueue](#) on page 539

[*LookupTopicConnectionFactory](#) on page 539

[*LookupXATopicConnectionFactory](#) on page 540

- *[LookupTopic](#) on page 541
- *[CreateXid](#) on page 541
- *[LookupXADataSource](#) on page 542
- *[LookupQueueConnectionFactoryExt](#) on page 542
- *[LookupXAQueueConnectionFactoryExt](#) on page 543
- *[LookupXATopicQueueConnectionFactoryExt](#) on page 544

*LookupQueueConnectionFactory

Syntax

```
*LookupQueueConnectionFactory(dllname, initString, hostname, port,  
usPortOffset, iMaxRetires)
```

Description

Constructs a QueueConnectionFactory for the specified host and port using the given <iniString> name. You can replace this method if you are using your own naming service lookup API call.

Parameters

Name	Type	Description
dllname	const char*	dllname should always be set to MSCLIENT_DLL_NAME unless you rename the default libraries shipped with the API Kit.
initString	const char*	User defined name.
hostName	const char*	The default is DEFAULT_SERVER_NAME if you do not define a name.
port	unsigned short	The default is DEFAULT_PORT if you do not define a port.
usPortOffset	unsigned short	The default port offset is 0 if you do not define an offset.
iMaxRetries	int	The default number of maximum retries is 30 if you do not define a maximum number of retries.

Returns

QueueConnectionFactory

*LookupXAQueueConnectionFactory

Syntax

```
*LookupXAQueueConnectionFactory(dllname, initString, hostname, port,  
usPortOffset, iMaxRetires)
```

Description

Constructs a QueueXAConnectionFactory for the specified host and port using the given <iniString> name. You can replace this method if you are using your own naming service lookup API call.

Parameters

Name	Type	Description
dllname	const char*	dllname should always be set to MSCLIENT_DLL_NAME unless you rename the default libraries shipped with the API Kit.
initString	const char*	User defined name.
hostName	const char*	The default is DEFAULT_SERVER_NAME if you do not define a name.
port	unsigned short	The default is DEFAULT_PORT if you do not define a port.
usPortOffset	unsigned short	The default port offset is 0 if you do not define an offset.
iMaxRetries	int	The default number of maximum retries is 30 if you do not define a maximum number of retries.

Returns

XAQueueConnectionFactory

*LookupQueue

Syntax

```
*LookupQueue(const char*, const char*)
```

Description

Constructs a topic using the given topicName.

Parameters

Name	Type	Description
User defined	const char*	User defined string.
User defined	const char*	User defined string.

Returns

Queue

*LookupTopicConnectionFactory

Syntax

```
*LookupTopicConnectionFactory(dllname, initString, hostname, port,
usPortOffset, iMaxRetires)
```

Description

Constructs a TopicConnectionFactory for the specified host and port using the given <iniString> name. You can replace this method if you are using your own naming service lookup API call.

Parameters

Name	Type	Description
dllname	const char*	dllname should always be set to MSCCLIENT_DLL_NAME unless you rename the default libraries shipped with the API Kit.
initString	const char*	User defined name.
hostName	const char*	The default is DEFAULT_SERVER_NAME if you do not define a name.
port	unsigned short	The default is DEFAULT_PORT if you do not define a port.
usPortOffset	unsigned short	The default port offset is 0 if you do not define an offset.
iMaxRetries	int	The default number of maximum retries is 30 if you do not define a maximum number of retries.

Returns

TopicConnectionFactory

*LookupXATopicConnectionFactory

Syntax

```
*LookupXATopicConnectionFactory(dllname, initString, hostname, port,
usPortOffset, iMaxRetires)
```

Description

Constructs a XATopicConnectionFactory for the specified host and port using the given <iniString> name. You can replace this method if you are using your own naming service lookup API call.

Parameters

Name	Type	Description
dllname	const char*	dllname should always be set to MSCLIENT_DLL_NAME unless you rename the default libraries shipped with the API Kit.
initString	const char*	User defined name.
hostName	const char*	The default is DEFAULT_SERVER_NAME if you do not define a name.
port	unsigned short	The default is DEFAULT_PORT if you do not define a port.
usPortOffset	unsigned short	The default port offset is 0 if you do not define an offset.
iMaxRetries	int	The default number of maximum retries is 30 if you do not define a maximum number of retries.

Returns

XATopicConnectionFactory

*LookupTopic

Syntax

```
*LookupTopic(dllname, topicName)
```

Description

Constructs a topic using the given topicName.

Parameters

Name	Type	Description
dllname	const char*	dllname should always be set to MSCLIENT_DLL_NAME unless you rename the default libraries shipped with the API Kit.
topicName	const char*	Name of the topic.

Returns

Topic

*CreateXid

Syntax

```
*CreateXid(dllname)
```

Description

Constructs an Xid.

Parameters

Name	Type	Description
dllname	const char*	dllname should always be set to MSCLIENT_DLL_NAME unless you rename the default libraries shipped with the API Kit.

Returns

Xid

*LookupXADataSource

Syntax

```
*LookupXADataSource(const char*, const char*)
```

Description

Used to retrieve the XADataSource using a thirdparty library. For example, call LookupXADataSource(ORACLE_DLL_Name, "myname") to get and instance of XADataSource

Parameters

Name	Type	Description
User defined	const char*	User defined String.
User defined	const char*	User defined String.

Returns

XADataSource

*LookupQueueConnectionFactoryExt

Syntax

```
*LookupQueueConnectionFactoryExt(dllname, initString, hostname, port, usPortOffset, iMaxRetires, iInterval)
```

Description

Constructs QueueConnectionFactoryExt for the specified host and port using the given <iniString> name. You can replace this method if you are using your own naming service lookup API call.

Parameters

Name	Type	Description
dllname	const char*	dllname should always be set to MSCLIENT_DLL_NAME unless you rename the default libraries shipped with the API Kit.
initString	const char*	User defined name.
hostName	const char*	The default is DEFAULT_SERVER_NAME if you do not define a name.
port	unsigned short	The default is DEFAULT_PORT if you do not define a port.
usPortOffset	unsigned short	The default port offset is 0 if you do not define an offset.
iMaxRetries	int	The default number of maximum retries is 30 if you do not define a maximum number of retries.
iInterval	int	The default retry interval is 100 millisecond if you do not define the retry interval.

Returns

QueueConnectionFactory

*LookupXAQueueConnectionFactoryExt

Syntax

```
*LookupXAQueueConnectionFactoryExt(dllname, initString, hostname,  
port, usPortOffset, iMaxRetires, iInterval)
```

Description

Constructs XAQueueConnectionFactoryExt for the specified host and port using the given <iniString> name. You can replace this method if you are using your own naming service lookup API call.

Parameters

Name	Type	Description
dllname	const char*	dllname should always be set to MSCLIENT_DLL_NAME unless you rename the default libraries shipped with the API Kit.
initString	const char*	User defined name.
hostName	const char*	The default is DEFAULT_SERVER_NAME if you do not define a name.
port	unsigned short	The default is DEFAULT_PORT if you do not define a port.
usPortOffset	unsigned short	The default port offset is 0 if you do not define an offset.
iMaxRetries	int	The default number of maximum retries is 30 if you do not define a maximum number of retries.
iInterval	int	The default retry interval is 100 millisecond if you do not define the retry interval.

Returns

XAQueueConnectionFactory

*LookupXATopicQueueConnectionFactoryExt

Syntax

```
*LookupXATopicQueueConnectionFactoryExt(dllname, initString, hostname,  
port, usPortOffset, iMaxRetries, iInterval)
```

Description

Constructs XATopicQueueConnectionFactoryExt for the specified host and port using the given <initString> name. You can replace this method if you are using your own naming service lookup API call.

Parameters

Name	Type	Description
dllname	const char*	dllname should always be set to MSCLIENT_DLL_NAME unless you rename the default libraries shipped with the API Kit.
initString	const char*	User defined name.
hostName	const char*	The default is DEFAULT_SERVER_NAME if you do not define a name.
port	unsigned short	The default is DEFAULT_PORT if you do not define a port.
usPortOffset	unsigned short	The default port offset is 0 if you do not define an offset.
iMaxRetries	int	The default number of maximum retries is 30 if you do not define a maximum number of retries.
iInterval	int	The default retry interval is 100 millisecond if you do not define the retry interval.

Returns

XATopicQueueConnectionFactory

Configuring the Multiplexer e*Way

6.1 Configuring the Multiplexer Client

After the e*Gate API Kit is installed, 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.

This section provides steps for setting up the Multiplexer and Muxpooler.

6.1.1. Before You Begin

To allow 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 you will specify in [“Push IP Port” on page 548](#).

6.1.2. Setting up the Multiplexer

To begin using the Multiplexer, do the following:

- 1 Copy the following files from your `<eGate>\client\bin` folder to a folder on your external system:
 - ♦ `stc_xipmpclnt.dll`
 - ♦ `stc_common.dll`
 - ♦ `stc_ewipmpclnt.dll`
- 2 From the command prompt of the external system, register all three files into Windows Registry by using the following command:

```
regsvr32 <path>\<filename>
```

6.1.3. Setting up the Muxpooler

To begin using the Muxpooler, do the following:

- 1 Copy the following files from your `<eGate>\client\bin` folder to a folder on your external system:
 - ♦ `stc_common.dll`
 - ♦ `stcph.jar`
- 2 Be sure that **stcph.jar** is included in your classpath and then, from a command prompt of the external system, register **stc_common.dll** in the Windows Registry with the following command:

```
regsvr32 <path>\stc_common.dll
```

This remainder of 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`. Whenever this guide discusses procedures within the context of an e*Gate GUI (such as 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.*

6.2 Configuring the Multiplexer Server

6.2.1. 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 Schema Designer’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 contained in a single section: **General Settings**.

General Settings

Request Reply IP Port

Description

Specifies the IP port that the e*Way will listen (bind) to 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

If an Event received by this e*Way has **0** for all fields in the 24-byte MUX header, it is sent to all callers of the **WaitForUnsolicited**. This parameter is optional. If set to **0**, the e*Way follows the request/reply scenario and does not accept unsolicited Events.

Rollback if no Clients on Push Port

Description

Specifies whether Events continually roll back if no push clients are connected.

Required Values

Yes or **No**. If set to **Yes**, Events continually roll back if no push client is connected.

Wait For IQ Ack

Description

Specifies whether the send client function is to wait for an acknowledgment (that the Event was committed to the IQ) before returning.

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: *The **Wait for IQ Ack** parameter should be set to **Yes** only if the data must be committed to the IQ on every transaction before the API returns to the client. A setting of **Yes** causes significant performance impact. 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 Event (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 Event when an external client disconnects.

MUX Instance ID

Description

Specifies an 8-byte header to prepend to the 24-byte session ID of an Event received from an external connection before sending to e*Gate; see [“Notes on Session ID, Instance ID, and Recovery ID” on page 549](#).

Required Values

A string. The default is **0**. If set to a value other than **0**, the first eight bytes are prepended to the 24-byte session ID when the Event is sent to e*Gate.

*Note: The strings **00**, **000**, and so forth (up to **00000000**) are valid, although inadvisable. However, the single-byte string **0** has a special meaning: “Do not use this option.”*

MUX Recovery ID

Description

Specifies an 8-byte header to prepend to the 24-byte session ID of an Event to republish back to e*Gate if the multiplexer finds that the session has been dropped.

Required Values

A string. The default is **0**. If set to a value other than **0**, the first eight bytes are prepended to the 24-byte session ID when the Event is republished to e*Gate.

6.2.2. Notes on Session ID, Instance ID, and Recovery ID

When an external system connects to the multiplexer, the e*Way prepends a 24-byte header, called the *session ID*, to each Event before sending it into e*Gate. e*Gate then performs the processing and returns the result to the multiplexer, which expects this 24-byte header to remain prepended to the Event after it is returned from e*Gate.

However, because many implementations involve more than one multiplexer, there can be a need to know which one of the multiplexers generated the request—for example, to allow a single Collaboration Rule to process or route Events from different multiplexers, to recover undelivered Events due to dropped sessions, or even to facilitate a second delivery channel for the undelivered Events. To allow for such interfaces, the multiplexer can be configured to prepend a *further* 8-byte header before the session ID on Events it sends into e*Gate. This 8-byte header is the value specified by the configuration parameter **MUX Instance ID**. If no value is specified for this parameter, only the 24-byte session ID is prepended. Upon successful back-end processing, e*Gate does *not* return the MUX Instance ID; it returns only the session ID.

If the multiplexer receiving this Event finds that the corresponding session has been dropped, it can republish the Event into e*Gate, prepending a different 8-byte header. This “backup” header is the value specified by the configuration parameter **MUX Recovery ID**. If no value is specified for this parameter, the multiplexer can drop the Event if the session has been dropped.

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 as 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.

7.1 Implementing the Multiplexer Models

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

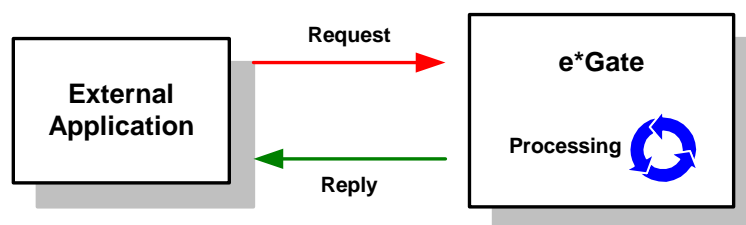
- 1 Request/Reply, where data is sent to the e*Gate system and a response is returned
- 2 Send-only, where data is sent to the e*Gate system but no data is returned
- 3 Receive, where an external system connects to the e*Gate system and allows for the delivery unsolicited Events

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

7.1.1. Multiplexer Overview

Request Reply

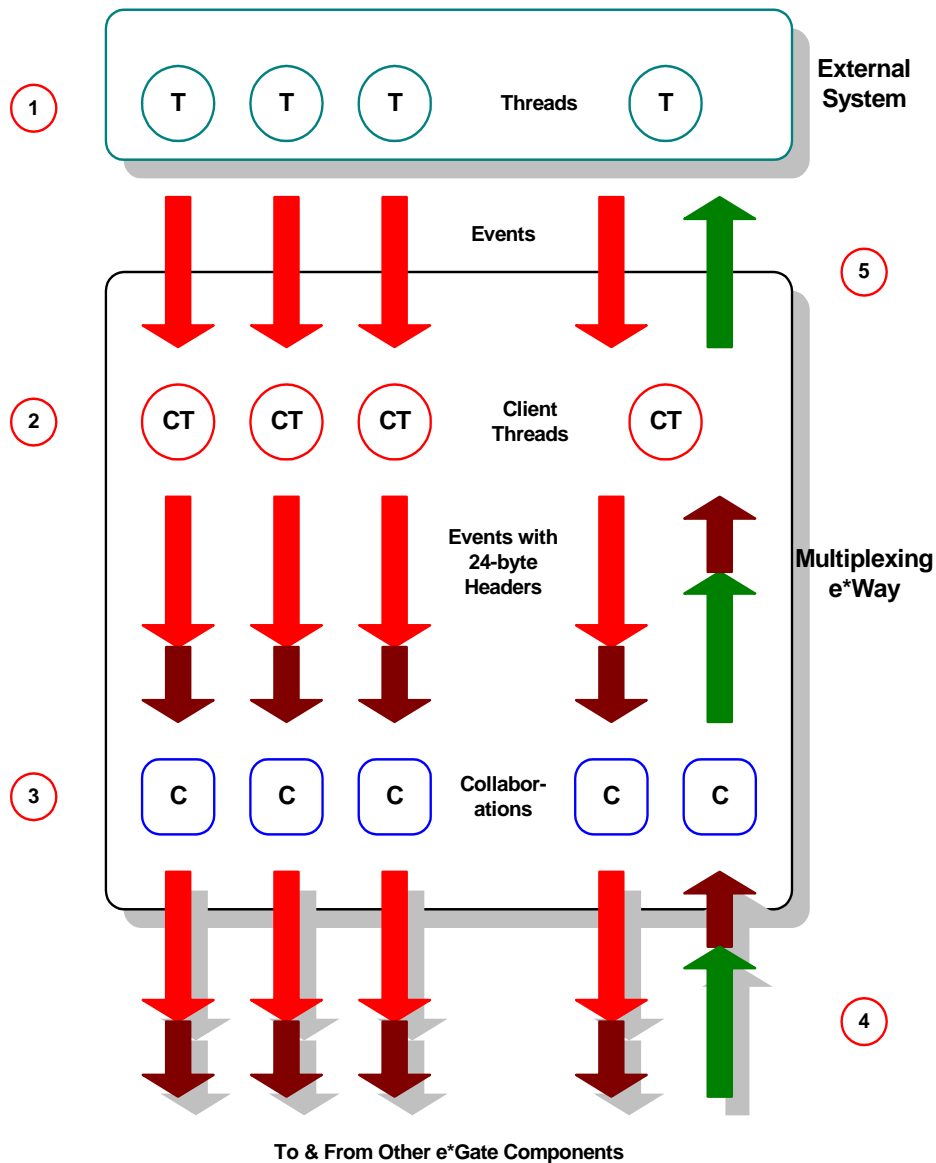
Figure 35 The Multiplexer concept



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

Figure 36 illustrates how the multiplexing e*Way receives data from an external application and returns processed data to the same application.

Figure 36 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.

7.1.2. Multiplexer Request/Reply Sample Schema

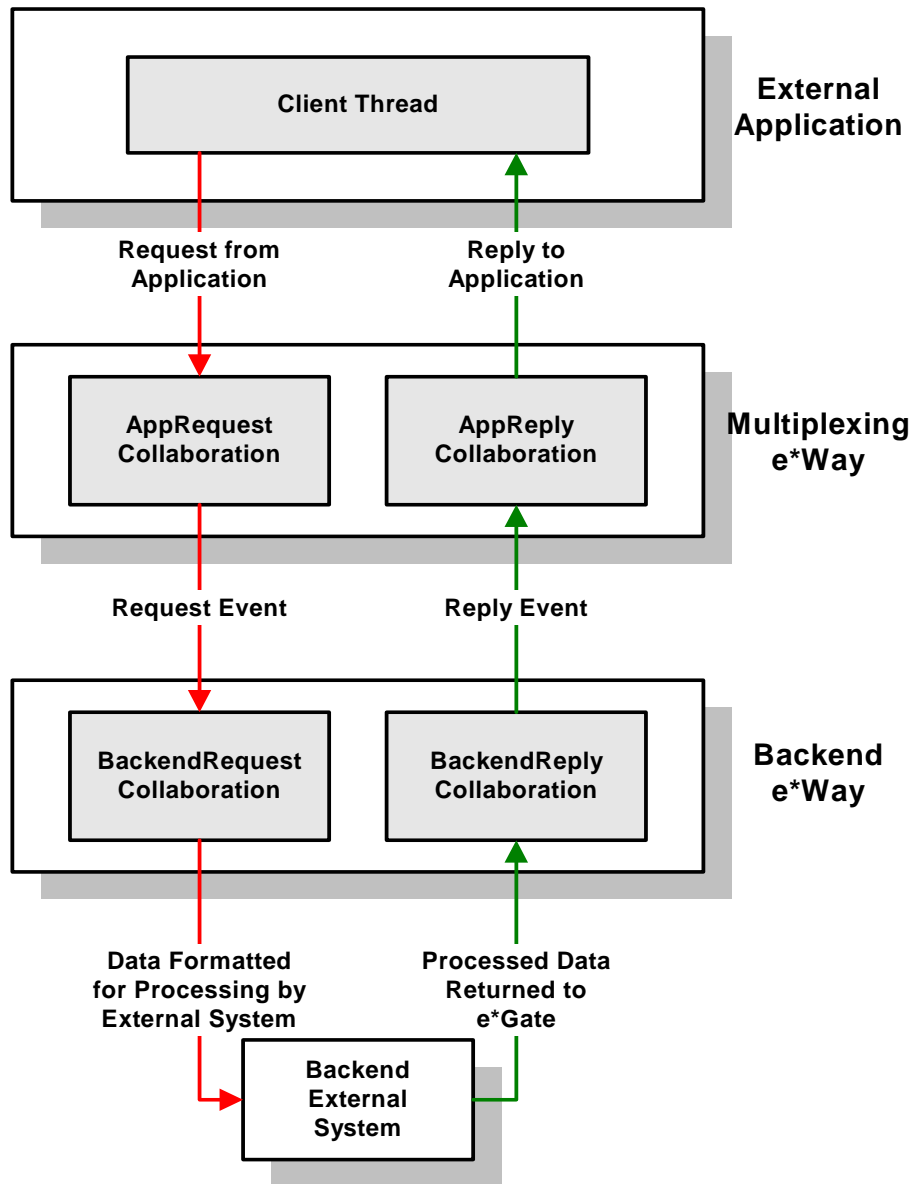
Request/Reply schemas have two classes of components:

- 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 37 below illustrates a Request/Reply schema.

Figure 37 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.

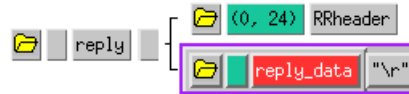
- 5 The **Reply** Collaboration within the multiplexing e*Way subscribes to the Reply Event.
- 6 The multiplexing e*Way returns the processed data to the requesting thread in the external application.

7.1.3. ETDs, Collaboration Rules, and the “Return Address” Header

As discussed in “Request/Reply” on page 31, 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.

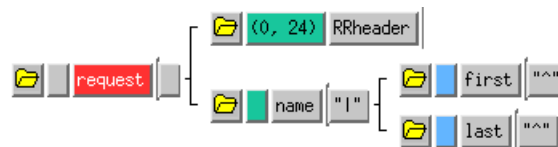
The simplest Event Type Definition (ETD) that can be used within a Request/Reply schema has two nodes: one for the header, the second for the remainder of the Event data.

Figure 38 The simplest Request/Reply ETD



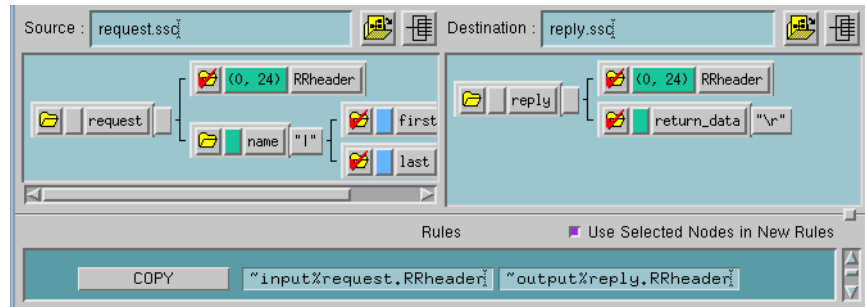
This ETD is sufficient if you wish to send data through the e*Gate system simply as a blob. If your data has a more complex structure, add subnodes to the “data” node, then describe the structure of the data within those subnodes. Figure 38 below illustrates an ETD that describes delimited data (for example, as in the data “First name^Last name”).

Figure 39 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 40 below).

Figure 40 Copying the Request/Reply header



7.1.4. Using the C APIs

The C application must do the following:

- 1 Load the following header files, located on the installation CD-ROM in the root `\sdk` directory:
`gendefs.h, tracelog.h, ewipmpclnt.h`
- 2 Link the `stc_ewipmpclnt.dll` and the `stc_common.dll` libraries at compile time.
- 3 Use the `EWIPMP_Open` function (see [“EWIPMP_Open” on page 562](#)) 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 (see [“EWIPMP_Send” on page 563](#)) to send data to the e*Gate system.
- 7 Use the `EWIPMP_Wait` function (see [“EWIPMP_Wait” on page 563](#)) to cause execution to pause long enough for e*Gate to process and return the data.
- 8 Use the `EWIPMP_Free` function (see [“EWIPMP_Free” on page 561](#)) to free the memory associated with the returned data in the message buffer.
- 9 Use the `EWIPMP_Close` function (see [“EWIPMP_Close” on page 560](#)) to close the connection.

7.1.5. 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 Use the various `set` methods (described beginning with [“setDebug” on page 580](#)) to define the host name, TCP/IP port, expiration time, and timeout.
- 4 Use the `connect` method (see [“connect” on page 575](#)) 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 579](#)) to send the request to the e*Gate system.
- 8 Use one of the **getResponse** methods (such as [“getResponse” on page 577](#)) to retrieve the response from the e*Gate system.
- 9 Use the **disconnect** method (see [“disconnect” on page 575](#)) to close the connection.

A commented **sample.java** file is available on the e*Gate installation CD-ROM; for more information, see [“Sample Implementation” on page 559](#).

7.1.6. Using the ActiveX Control Within Visual Basic 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 570](#)) 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 573](#)) to send data to the e*Gate system.
- 7 Use the **Wait** method (see [“Wait” on page 573](#)) 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 572](#)) to display the returned data.
- 9 Use one of the **LastError** methods (such as [“LastErrorText” on page 571](#)) to handle errors.
- 10 Use the **Disconnect** method (see [“Disconnect” on page 570](#)) to close the connection.

Additional information can be found in commented sample files; for more information, see [“Sample Implementation” on page 559](#).

7.1.7. Using Perl APIs

The Perl script must do the following:

- 1 Use the **Multiplexer_Init** subroutine (see [“Multiplexer_Init” on page 592](#)) 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 593](#)) to open a connection to the e*Gate Participating Host.

- 5 Use the **Multiplexer_Send** subroutine (see “**Multiplexer_Send**” on page 594) to send data to the e*Gate Participating Host.
- 6 Use the **Multiplexer_Wait** subroutine (see “**Multiplexer_Wait**” on page 595) 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 594) to obtain the returned data and display it within the user’s browser.
- 8 Use the **Multiplexer_Free** subroutine (see “**Multiplexer_Free**” on page 592) to free the memory associated with the returned data.
- 9 Use the **Multiplexer_Close** subroutine (see “**Multiplexer_Close**” on page 591) to close the connection.

7.2 Using the COBOL APIs

The following code demonstrates a sample set of actions. The string **MUXxxx** means: MUXAPI (for CICS); MUXIMS (for IMS); or MUXBAT (for Batch).

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

Note: *If the data you are sending is larger than 32KB, you need to break the data up into smaller packets adding a unique identifier to each. To identify the data packets and assure their delivery, assign each packet a unique identifier. Assign each packet a sequence number and the total number of data packets to the first message. By breaking up the message into individual packets, ELS can reassemble the packets into the complete message. An algorithm needs to be developed for splitting the original message, coordinating the number of data packets and reconstituting the original message on the client side.*

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

MUXxxx means: MUXAPI (for CICS); MUXIMS (for IMS); or 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
```

7.3 Sample Implementation

A sample implementation is available in the **samples** directory of the e*Gate CD-ROM. Follow the directions in the **samples\ewmux\Readme.txt** file.

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 Schema Designer 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 Oracle for more information.*

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 560
- [COBOL APIs](#) on page 565
- [ActiveX APIs](#) on page 569
- [Java Methods](#) on page 574
 - ♦ [com.stc.MUXPooler](#) on page 584
- [Perl Subroutines](#) on page 591

8.1 C API Function Prototypes

The file `ewipmpclnt.h` defines the following function prototypes:

- [EWIPMP_Close](#) on page 560
- [EWIPMP_Free](#) on page 561
- [EWIPMP_Open](#) on page 562
- [EWIPMP_Send](#) on page 563
- [EWIPMP_Wait](#) on page 563

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, pszServerHost, 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.
pszServerHost	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 560) 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 561).
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_ALLOCATIO N	0x20001000

Examples

```
// wait for reply
//
if (!EWIPMP_Wait(hIPMP, &dwMessageLength, (BYTE**) &pBuffer,
                cMillisecondsToWait, 0, NULL))
{
    goto WaitFailed;
}
```

8.2 COBOL APIs

- **Open** on page 565
- **Send** on page 566
- **Receive** on page 567
- **Close** on page 569

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 and returns 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 (see below), **MUXAPI-errno** contains an error number.

MUXAPI - retcode

A 4-byte signed binary number, initialized to zero.

Returns

A value of zero or greater indicates a successful call. A negative value signifies an error; the error number is contained in **MUXAPI-errno**.

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 and returns 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 acknowledgment (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 (see below), **MUXAPI-errno** contains an error number.

MUXAPI - retcode

A 4-byte signed binary number, initialized to zero.

Returns

A value of zero or greater indicates a successful call. A negative value signifies an error; the error number is contained in **MUXAPI-errno**.

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 and returns 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 (see below), **MUXAPI-errno** contains an error number.

MUXAPI - retcode

A 4-byte signed binary number, initialized to zero.

Returns

A value of zero or greater indicates a successful call. A negative value signifies an error; the error number is contained in **MUXAPI-errno**.

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.
```

Parameters and returns set by the application

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 (see below), **MUXAPI-errno** contains an error number.

MUXAPI - retcode

A 4-byte signed binary number, initialized to zero.

Returns

A value of zero or greater indicates a successful call. A negative value signifies an error; the error number is contained in **MUXAPI-errno**.

8.3 ActiveX APIs

The e*Gate API Kit ActiveX control supports the following methods:

- **Connect** on page 570
- **Disconnect** on page 570
- **LastErrorCode** on page 571
- **LastErrorText** on page 571

- [ReplyMessageAsArray](#) on page 572
- [ReplyMessageAsString](#) on page 572
- [ReplyMessageSize](#) on page 572
- [Send](#) on page 573
- [Wait](#) on page 573

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

```
Disconnect
```

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 e*Gate 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:

- **connect** on page 575
- **disconnect** on page 575
- **getHost** on page 576
- **getPort** on page 576
- **getResponse** on page 577
- **getResponseBytes** on page 577
- **getSecondsToExpire** on page 578
- **getSleepDuration** on page 578
- **getTimeout** on page 579
- **sendMessage** on page 579
- **setDebug** on page 580
- **setHost** on page 581
- **setPort** on page 581
- **setSecondsToExpire** on page 582
- **setSleepDuration** on page 582
- **setTimeout** on page 583

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.

Variable	Type	Value	Meaning
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

java.lang.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

java.lang.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 577, 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 574](#) 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();
```

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 574](#) 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

```
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(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	Set this to true to enable debugging methods. The default, false , suppresses debugging.

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 574](#) for more information.

Type

void

Parameters

Name	Type	Description
seconds	long	The new expiration time.

Return Values

None.

Examples

```
mux.setSecondsToExpire(10);
```

setSleepDuration

Syntax

```
setSleepDuration(milliseconds)
```

Description

setSleepDuration changes the current internal sleep interval for MUX reply waiting.

Type

void

Parameters

Name	Type	Description
milliseconds	long	The time (in milliseconds) that MUX waits for each phase of reply processing.

Return Values

None.

Examples

```
long sleepduration = mux.setSleepDuration(1000);
```

setTimeout

Syntax

```
setTimeout(milliseconds)
```

Description

setTimeout sets the timeout to the specified value. See [“Defaults” on page 574](#) 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

The **MUXPooler** class has three Constructors for instantiating an object:

- `public MUXPooler()` : Instantiates the object only. Each of the additional attributes must be called individually.
 - ♦ [setHost](#) on page 589
 - ♦ [setPort](#) on page 590
 - ♦ [setTimeout](#) on page 590
 - ♦ [setSecondsToExpire](#) on page 590
- `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 585
- [disconnect](#) on page 585
- [disconnect](#) on page 585
- [getConnectionCount](#) on page 586
- [getHost](#) on page 586
- [getPort](#) on page 587
- [getSecondsToExpire](#) on page 587
- [getTimeout](#) on page 587

- [resizeMUXPool](#) on page 588
- [sendBytes](#) on page 588
- [sendMessage](#) on page 588
- [setConnectionCount](#) on page 589
- [setHost](#) on page 589
- [setPort](#) on page 590
- [setSecondsToExpire](#) on page 590

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 milliseconds.

getTimeout

Syntax

```
getTimeout()
```

Description

getTimeout returns the number of milliseconds to wait for a response before timeout.

Parameters

None.

Return Values

integer

Returns the number of milliseconds.

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

None.

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

None.

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

None.

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

None.

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

None.

8.6 Perl Subroutines

The e*Gate API Kit supports the following Perl extension subroutines:

- [Connect](#) on page 570
- [LastErrorCode](#) on page 571
- [LastErrorText](#) on page 571
- [Multiplexer_Close](#) on page 591
- [Multiplexer_Close](#) on page 591
- [Multiplexer_Free](#) on page 592
- [Multiplexer_Init](#) on page 592
- [Multiplexer_Open](#) on page 593
- [Multiplexer_Send](#) on page 594
- [Multiplexer_ToString](#) on page 594
- [Multiplexer_Wait](#) on page 595
- [ReplyMessageAsArray](#) on page 572
- [Send](#) on page 573
- [Wait](#) on page 573

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 ("xxx/xxx/xxx"); Where xxx represents your file location.

use CGI qw/:standard/;
use stc_ewipmpclntperl.dll;

# call Multiplexer_Init to define the dll path
Multiplexer_Init("xxx/xxx/xxx"); Where xxx represents your file
location.
```

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 multiplexing e*Way. If this value is 0 (zero), the default port 26051 is used.
flags	integer	Command flags. Always set to 0 (zero) unless directed otherwise by Oracle support personnel.
reserved	integer	Reserved for future use. Always set to 0 (zero) unless directed otherwise by Oracle 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 unless directed otherwise by Oracle support personnel.
reserved	integer	Reserved for future use. Always set to 0 (zero) unless directed otherwise by Oracle 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 571 for more information.

Multiplexer_Wait

Syntax

```
Multiplexer_Wait(handle, message-length, millisecond-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 Oracle support personnel.
reserved	integer	Reserved for future use. Always set to 0 (zero) unless directed otherwise by Oracle 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.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	Something other than an acknowledgment (ACK) was received in response to a SEND.	Internal error. Call your system administrator.
3004	Timed out waiting for an ACK after a SEND.	Increase the value of MUXAPI-hsecs-for-ack .
3005	Timed out waiting for a RECEIVE.	Increase the value of MUXAPI-hsecs-to-wait . Note: The units for timeout values are hundredths of a second. For example, to set the timeout value to one second, pass the value 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 SETIBMPT (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 ensuring 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 Setsockopt() 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 too 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	An SRMOUNT 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	EIBMUCVERR	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	EIMBADTCPNAME	All	A TCP/IP name that is not valid was detected.	Correct the name specified in the IBM_TCPIIMAGE structure.
1011	EIMBADTCPNAME	Setibmopt	A TCP/IP name that is not valid was detected.	Correct the name specified in the IBM_TCPIIMAGE.
1011	EIMBADTCPNAME	INITAPI	A TCP/IP name that is not valid was detected.	Correct the name specification the IDENT option TCPNAME field.
1012	EIMBADREQUESTCODE	All	A request code that is not valid was detected.	Contact your system administrator.
1013	EIMBADCONNECTIONSTATE	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	EIMBADCONNECTIONMATCH	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	EIBMINVTSRUSERDATA	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 INITIALIZE 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.

Error Code	Problem Description	System Action	Programmer's Response
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.
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.

Error Code	Problem Description	System Action	Programmer's Response
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.
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.

Error Code	Problem Description	System Action	Programmer's Response
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.
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.

Error Code	Problem Description	System Action	Programmer's Response
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

A

Acknowledge
 C function for JMS 362
 acknowledge
 C ++ function for JMS 459
 C++ function for JMS 459
 C++ functions for JMS 459
 Acknowledge Method 292, 300, 308, 324, 332
 ActiveX APIs for MUX
 Connect 570
 Disconnect 570
 LastErrorCode 571
 LastErrorText 571
 ReplyMessageAsArray 572
 ReplyMessageAsString 572
 ReplyMessageSize 572
 Send 573
 Wait 573
 ActiveX control
 Class ID 574
 in Visual Basic applications 556
 AutoAcknowledge mode 359

B

bytesMessage
 C++ function for JMS 503
 BytesMessage Object 292
 BytesMessage Property 310, 333

C

C API for JMS 355
 (diagrammed) 356
 C APIs for MUX
 EWIPMP_Close 560
 EWIPMP_Free 561
 EWIPMP_Open 562
 EWIPMP_Send 563
 EWIPMP_Wait 563
 C functions for JMS
 Acknowledge 362
 CharToWString 452
 ClearBody 363
 ClearProperties 363
 ConnectionClose 397
 ConnectionCreateSession 399
 ConnectionCreateTopicSession 399
 ConnectionGetClientID 397
 ConnectionSetClientID 397
 ConnectionStart 398
 ConnectionStop 398
 CreateQueueConnection 396, 507
 CreateQueueConnectionFactory 395
 CreateQueueRequestor 444
 CreateTopicConnection 412
 CreateTopicConnectionFactory 412
 CreateTopicRequestor 442
 DeleteDestination 414
 DeleteMessage 452
 DeleteQueueConnection 447
 DeleteQueueConnectionFactory 447
 DeleteQueueReceiver 448
 DeleteQueueRequestor 448
 DeleteQueueSender 448
 DeleteQueueSession 449
 DeleteTopicConnection 450
 DeleteTopicConnectionFactory 449
 DeleteTopicPublisher 451
 DeleteTopicRequestor 451
 DeleteTopicSession 450
 DeleteTopicSubscriber 450
 DeleteWString 453
 DeleteWStringList 454
 DestinationToString 414
 GetBooleanProperty 364
 GetByteProperty 365
 GetDestinationName 413
 GetDoubleProperty 365
 GetFloatProperty 366
 GetIntProperty 366
 GetJMS_ProducerID 456
 GetJMSCorrelationID 372
 GetJMSCorrelationIDAsBytes 372
 GetJMSDeliveryMode 373
 GetJMSExpiration 373
 GetJMSMessageID 374
 GetJMSPriority 374
 GetJMSRedelivered 375
 GetJMSReplyTo 375
 GetJMSTimestamp 376
 GetJMSType 376
 GetLongProperty 367
 GetMessageType 381
 GetPropertyName 454
 GetShortProperty 367
 GetStringProperty 368
 GetText 394

PropertyExists 364
 QueueReceiverClose 415
 QueueReceiverGetMessageSelector 416
 QueueReceiverGetQueue 417
 QueueReceiverReceive 416
 QueueReceiverReceiveNoWait 417
 QueueReceiverReceiveTimeout 417
 QueueRequestorClose 445
 QueueRequestorRequest 445
 QueueRequestorRequestTimeout 446
 QueueSenderClose 422
 QueueSenderGetDeliveryMode 423, 513
 QueueSenderGetDisableMessageID 423
 QueueSenderGetDisableMessageTimestamp 424
 QueueSenderGetJMS_ProducerID 424
 QueueSenderGetPriority 424
 QueueSenderGetQueue 425
 QueueSenderGetTimeToLive 425
 QueueSenderSend 426
 QueueSenderSendEx 426
 QueueSenderSendToQueue 427
 QueueSenderSendToQueueEx 428
 QueueSenderSetDeliveryMode 429
 QueueSenderSetDisableMessageID 429
 QueueSenderSetDisableMessageTimestamp 430
 QueueSenderSetJMS_ProducerID 430
 QueueSenderSetPriority 431
 QueueSenderSetTimeToLive 431
 QueueSessionClose 401
 QueueSessionCreateQueue 404
 ReadBoolean 383
 ReadByte 383
 ReadBytes 384, 456
 ReadChar 384
 ReadDouble 385
 ReadFloat 385
 ReadInt 386
 ReadLong 386
 ReadShort 386
 ReadUnsignedByte 387
 ReadUnsignedShort 387
 ReadUTF 388
 Reset 388
 SessionCommit 401
 SessionCreateBytesMessage 403
 SessionCreateDurableSubscriber 407
 SessionCreateDurableSubscriberMessageSelector 408
 SessionCreatePublisher 408
 SessionCreateReceiveMessageSelector 405
 SessionCreateReceiver 405
 SessionCreateSender 406
 SessionCreateSubscriber 409
 SessionCreateSubscriberMessageSelector 409
 SessionCreateTemporary 406
 SessionCreateTextMessage 403
 SessionCreateTextMessageEx 404
 SessionGetTransacted 402
 SessionRecover 402
 SessionRollback 402
 SessionUnsubscribe 411
 SetBooleanProperty 368
 SetByteProperty 369
 SetDestinationName 414
 SetDoubleProperty 369
 SetFloatProperty 370
 SetIntProperty 370
 SetJMS_ProducerID 456
 SetJMSCorrelationID 377
 SetJMSCorrelationIDAsBytes 377
 SetJMSDeliveryMode 378
 SetJMSExpiration 378
 SetJMSMessageID 378
 SetJMSPriority 379
 SetJMSRedelivered 379
 SetJMSReplyTo 380
 SetJMSTimestamp 380
 SetJMSType 381
 SetLongProperty 371
 SetShortProperty 371
 SetStringProperty 372
 SetText 395
 TopicPublisherClose 432
 TopicPublisherGetDeliveryMode 433
 TopicPublisherGetDisableMessageID 433
 TopicPublisherGetDisableMessageTimestamp 434
 TopicPublisherGetJMS_ProducerID 434
 TopicPublisherGetPriority 434
 TopicPublisherGetTimeToLive 435
 TopicPublisherGetTopic 435
 TopicPublisherPublish 436
 TopicPublisherPublishEx 436
 TopicPublisherPublishToTopic 437, 524
 TopicPublisherPublishToTopicEx 438
 TopicPublisherSetDeliveryMode 438
 TopicPublisherSetDisableMessageID 439
 TopicPublisherSetDisableMessageTimestamp 439
 TopicPublisherSetJMS_ProducerID 440
 TopicPublisherSetPriority 440
 TopicPublisherSetTimeToLive 441
 TopicRequestorClose 443
 TopicRequestorRequest 442
 TopicRequestorRequestTimeout 443
 TopicSessionCreateTemporaryTopic 410
 TopicSessionCreateTopic 411

- TopicSubscriberClose 418
- TopicSubscriberGetMessageSelector 419
- TopicSubscriberGetNoLocal 419
- TopicSubscriberGetTopic 420
- TopicSubscriberReceive 420
- TopicSubscriberReceiveNoWait 421
- TopicSubscriberReceiveTimeout 420
- WriteBoolean 389
- WriteByte 389
- WriteBytes 390, 456
- WriteBytesEx 390
- WriteChar 391
- WriteDouble 391
- WriteFloat 392
- WriteInt 392
- WriteLong 392
- WriteShort 393
- WriteUTF 393
- WStringToChar 453
- C++ functions for JMS
 - acknowledge 459
 - readDouble 484
- CharToWString
 - C function for JMS 452
- Class ID, ActiveX control 574
- ClearBody
 - C function for JMS 363
- clearBody
 - C++ functions for JMS 459
- ClearBody Method 292, 301, 309, 324, 332
- ClearProperties
 - C function for JMS 363
- clearProperties
 - C++ function for JMS 460
- ClearProperties Method 292, 301, 309, 324, 332
- ClientAcknowledge mode 359
- ClientID Property 337, 351
- Close
 - COBOL API for MUX 569
- close
 - C++ function for JMS 499, 502, 505, 513
- Close Method 311, 344
- COBOL APIs for MUX
 - Close 569
 - Open 565
 - Receive 567
 - Send 566
- commit
 - C++ function for JMS 502, 533
- commit (transaction operation)
 - defined 359
- Commit Method 322, 340, 349, 353
- Configuration parameters
 - Push IP Port 548
- Request Reply IP Port 548
- Connect
 - ActiveX API for MUX 570
- connect
 - Java method for MUXPooler 585
- connection handle, subroutine to return 593
- Connection MetaData Object 300
- ConnectionClose
 - C function for JMS 397
- ConnectionCreateSession
 - C function for JMS 399
- ConnectionCreateTopicSession
 - C function for JMS 399
- ConnectionFactory Object 300
- ConnectionGetClientID
 - C function for JMS 397
- ConnectionSetClientID
 - C function for JMS 397
- ConnectionStart
 - C function for JMS 398
- ConnectionStop
 - C function for JMS 398
- CorrelationID Property 297, 306, 310, 329, 333
- CorrelationIDAsBytes Property 297, 306, 310, 329, 333
- Create 198, 199, 317, 340
- CreateBytesMessage Method 322, 341, 349, 353
- createDurableSubscriber
 - C++ function for JMS 528
- CreateDurableSubscriber Method 341
- CreateMapMessage Method 322, 341, 349, 353
- CreateMessage Method 320, 323, 341, 347, 349, 353
- createPublisher
 - C++ function for JMS 529
- CreatePublisher Method 341
- createQueue
 - C++ function for JMS 525
- CreateQueueConnection
 - C function for JMS 396, 507
- createQueueConnection
 - C++ function for JMS 507
- CreateQueueConnectionFactory
 - C function for JMS 395
- CreateQueueRequestor
 - C function for JMS 444
- createQueueSession
 - C++ function for JMS 501
- createReceiver
 - C++ function for JMS 526
- createSender
 - C++ function for JMS 527
- CreateStreamMessage 323
- CreateStreamMessage Method 342, 349, 353
- createSubscriber

- C++ function for JMS 529, 530
- CreateSubscriber Method 342
- createTemporaryQueue
 - C++ function for JMS 527
- createTemporaryTopic
 - C++ function for JMS 530
- CreateTemporaryTopic Method 342
- createTextMessage
 - C++ function for JMS 503, 504
- CreateTextMessage Method 323, 342, 349, 353
- createTopic
 - C++ function for JMS 531
- CreateTopic Method 342
- CreateTopicConnection
 - C function for JMS 412
- CreateTopicConnection Method 337, 352
- CreateTopicConnectionFactory
 - C function for JMS 412
- createTopicConnectionFactory
 - C++ function for JMS 508
- CreateTopicRequestor
 - C function for JMS 442
- createTopicSession
 - C++ function for JMS 504
- CreateTopicSession Method 336, 350, 504
- CreateXATopicConnection Method 352

D

- Delete
 - C++ function for JMS 510, 512
- Delete Method 331
- DeleteDestination
 - C function for JMS 414
- DeleteMessage
 - C function for JMS 452
- DeleteQueueConnection
 - C function for JMS 447
- DeleteQueueConnectionFactory
 - C function for JMS 447
- DeleteQueueReceiver
 - C function for JMS 448
- DeleteQueueRequestor
 - C function for JMS 448
- DeleteQueueSender
 - C function for JMS 448
- DeleteQueueSession
 - C function for JMS 449
- DeleteTopicConnection
 - C function for JMS 450
- DeleteTopicConnectionFactory
 - C function for JMS 449
- DeleteTopicPublisher
 - C function for JMS 451

- DeleteTopicRequestor
 - C function for JMS 451
- DeleteTopicSession
 - C function for JMS 450
- DeleteTopicSubscriber
 - C function for JMS 450
- DeleteWString
 - C function for JMS 453
- DeleteWStringList
 - C function for JMS 454
- delimited data, handling in ETDs 554
- DeliveryMode Property 297, 307, 313, 329, 338
- Destination Property 298, 307, 310, 329, 334
- DestinationToString
 - C function for JMS 414
- DisableMessageID Property 313, 339
- DisableMessageTimes Property 313
- DisableMessageTimestamp Property 339
- Disconnect
 - ActiveX API for MUX 570
- disconnect
 - Java method for MUX 575
 - Java method for MUXPooler 585
- DupsOKAcknowledge mode 359

E

- end
 - C++ function for JMS 537
- ETDs, sample 554
- Event Type Definitions, sample 554
- EWIPMP_Close function 560
- EWIPMP_Free function 561
- EWIPMP_Open function 562
- EWIPMP_Send function 563
- EWIPMP_Wait function 563
- Expiration Property 298, 307, 310, 330, 334

F

- files created by installation procedure 45

G

- getBoolean
 - C++ function for JMS 490
- GetBoolean Method 301, 490
- GetBooleanProperty
 - C function for JMS 364
- getBooleanProperty
 - C++ function for JMS 460, 468
- getBranchQualifier
 - C++ function for JMS 532

- getBytes
 - C++ function for JMS 491
- GetByte Method 301, 491
- GetByteProperty
 - C function for JMS 365
- getBytesProperty
 - C++ function for JMS 461, 468
- getBytes
 - C++ function for JMS 491
- GetBytes Methods 301, 491
- getChar
 - C++ function for JMS 492
- GetChar Property 301, 492
- getClientID
 - C++ function for JMS 500, 505
- getConnectionCount
 - Java method for MUXPooler 586
- GetDestinationName
 - C function for JMS 413
- getDisableMessageID
 - C++ function for JMS 513
- getDisableMessageTimestamp
 - C++ function for JMS 513
- getDouble
 - C++ function for JMS 492
- GetDouble Method 302, 492
- GetDoubleProperty
 - C function for JMS 365
- getDoubleProperty
 - C++ function for JMS 461, 469
- getExceptionListener
 - C++ function for JMS 506
- getfloat
 - C++ function for JMS 493
- GetFloat Method 302, 493
- GetFloatProperty
 - C function for JMS 366
- getFloatProperty
 - C++ function for JMS 461, 469
- getFormatId
 - C++ function for JMS 532
- getGlobalTransactionId
 - C++ function for JMS 532
- getHost
 - Java method for MUX 576
 - Java method for MUXPooler 586
- getInt
 - C++ function for JMS 493
- GetInt Method 302, 493
- GetIntProperty
 - C function for JMS 366
- getIntProperty
 - C++ function for JMS 462, 469
- GetJMS_ProducerID
 - C function for JMS 456
- getJMS_ProducerID
 - C++ function for JMS 514
- GetJMSCorrelationID
 - C function for JMS 372
- getJMSCorrelationID
 - C++ function for JMS 475
- GetJMSCorrelationIDAsBytes
 - C function for JMS 372
- getJMSCorrelationIDAsBytes
 - C++ function for JMS 475
- GetJMSDeliveryMode
 - C function for JMS 373
- getJMSDeliveryMode
 - C++ function for JMS 475
- GetJMSExpiration
 - C function for JMS 373
- getJMSExpiration
 - C++ function for JMS 476
- GetJMSMessageID
 - C function for JMS 374
- getJMSMessageID
 - C++ function for JMS 476
- GetJMSPriority
 - C function for JMS 374
- getJMSPriority
 - C++ function for JMS 476
- GetJMSRedelivered
 - C function for JMS 375
- getJMSRedelivered
 - C++ function for JMS 477
- GetJMSReplyTo
 - C function for JMS 375
- getJMSReplyTo
 - C++ function for JMS 477
- GetJMSTimestamp
 - C function for JMS 376
- getJMSTimestamp
 - C++ function for JMS 477
- GetJMSType
 - C function for JMS 376
- getJMSType
 - C++ function for JMS 477
- getlong
 - C++ function for JMS 493
- GetLong Method 302, 493
- GetLongProperty
 - C function for JMS 367
- getLongProperty
 - C++ function for JMS 462, 470
- GetMessageType
 - C function for JMS 381
- GetObject Method 302, 494
- getPort

- Java method for MUX 576
- Java method for MUXPooler 587
- getPriority
 - C++ function for JMS 514
- GetProperty 303, 494
- GetProperty Method 292, 309
- GetProperty Methods 332
- GetPropertyName
 - C function for JMS 454
- getPropertyName
 - C++ function for JMS 463
- getQueueName
 - C++ function for JMS 510
- getResponse
 - Java method for MUX 577
- getResponseBytes
 - Java method for MUX 577
- getSecondsToExpire
 - Java method for MUX 578
 - Java method for MUXPooler 587
- GetShort Method 303, 494
- GetShortProperty
 - C function for JMS 367
- getShortProperty
 - C++ function for JMS 463, 471
- getSleepDuration
 - Java method for MUX 578
- GetStringProperty
 - C function for JMS 368
- getStringProperty
 - C++ function for JMS 463, 471
- GetText
 - C function for JMS 394
- getText
 - C++ function for JMS 498
- getTimeout
 - Java method for MUX 579
 - Java method for MUXPooler 587
- getTimeToLive
 - C++ function for JMS 514
- getTopicName
 - C++ function for JMS 511
- getTransacted
 - C++ function for JMS 502
- getTransactionTimeout
 - C++ function for JMS 535

H

- handle, subroutine to return 593
- header, in Collaboration Rules 554
- HostName Property 337, 352

I

- installation
 - files/directories created 45
 - Windows 36
- Installing
 - OS/390 or z/OS 41
 - OS/400 39
 - UNIX 37, 38
 - Windows 36
- Instance ID, MUX 549
- isSameRM
 - C++ function for JMS 535
- ItemExists Method 303, 494

J

- Java methods for MUX
 - disconnect 575
 - getHost 576
 - getPort 576
 - getResponse 577
 - getResponseBytes 577
 - getSecondsToExpire 578
 - getSleepDuration 578
 - getTimeout 579
 - sendMessage 579
 - setDebug 580
 - setHost 581
 - setPort 581
 - setSecondsToExpire 582
 - setSleepDuration 582
 - setTimeout 583
- Java methods for MUXPooler
 - connect 585
 - disconnect 585
 - getConnectionCount 586
 - getHost 586
 - getPort 587
 - getSecondsToExpire 587
 - getTimeout 587
 - resizeMUXPool 588
 - sendBytes 588
 - sendMessage 588
 - setConnectionCount 589
 - setHost 589
 - setPort 590
 - setSecondsToExpire 590
 - setTimeout 590
- JMS
 - C API for 355
 - (diagrammed) 356
- JMS API in C
 - (diagram of object model) 356

- constants for 357, 360
- destructor functions for 446
- differences with Java API 456
- error codes and messages for 455
- helper interfaces for
 - WString 452
 - WStringList 454
- interfaces for
 - (listed) 360
 - BytesMessage 382
 - Destination 413
 - Message 361
 - Message, extended 381
 - QueueConnection 396
 - QueueConnectionFactory 395
 - QueueReceiver 415
 - QueueRequestor 444
 - QueueSender 421
 - QueueSession 400
 - TextMessage 394
 - TopicConnectionFactory 412
 - TopicPublisher 432, 521
 - TopicRequestor 442
 - TopicSubscriber 418
- structures for 357
- JMS COM APIs
 - Session Object Properties
 - MessageListener 323, 350
- JMS COM+ APIs
 - BytesMessage Object
 - ReadUnsignedShort Method 294
 - BytesMessage Object Methods
 - Acknowledge 292
 - ClearBody 292
 - ClearProperties 292
 - GetProperty 292
 - PropertyExists 293, 324
 - ReadBoolean 293
 - ReadByte 293
 - ReadBytes 293
 - ReadChar 293
 - ReadDouble 293
 - ReadFloat 293
 - ReadInt 294
 - ReadLong 294
 - ReadShort 294
 - ReadUnsignedByte 294
 - ReadUTF 294
 - Reset 294
 - SetBooleanProperty 294
 - WriteBoolean 295
 - WriteByte 295, 327
 - WriteBytes 295
 - WriteChar 295
 - WriteDouble 296
 - WriteFloat 296
 - WriteInt 296
 - WriteLong 296
 - WriteObject 296
 - WriteShort 297
 - WriteUTF 297
 - BytesMessage Object Properties
 - CorrelationID 297, 306
 - CorrelationIDAsBytes 297, 306
 - DeliveryMode 297, 329
 - Destination 298
 - Expiration 298, 307
 - MessageID 298, 330, 334
 - Priority 298, 334
 - Redelivered 298
 - ReplyTo 298, 308
 - Timestamp 299, 308
 - Type 299, 308
 - ClearMessage Object Methods
 - ClearProperties 301
 - Connection Object Methods
 - Start 299
 - Stop 299
 - ConnectionFactory Object Properties
 - Port 300
 - MapMessage Object Methods
 - Acknowledge 300
 - ClearBody 301
 - GetBoolean 301, 490
 - GetByte 301, 491
 - GetBytes 301, 491
 - GetChar 301, 492
 - GetDouble 302, 492
 - GetFloat 302, 493
 - GetInt 302, 493
 - GetLong 302, 493
 - GetObject 302, 494
 - GetProperty 303, 494
 - GetShort 303, 494
 - ItemExists 303, 494
 - PropertyExists 303, 495
 - SetBoolean 304, 495
 - SetByte 304, 495
 - SetBytes 304, 495
 - SetChar 304, 496
 - SetDouble 305, 496
 - SetFloat 305, 496
 - SetInt 305, 496
 - SetLong 305, 497
 - SetObject 305, 497
 - SetProperty 306, 497
 - SetShort 306, 497
 - SetString 306, 497

- MapMessage Object Properties
 - DeliveryMode 307
 - Destination 307
 - MapNames 307
 - MessageID 307
 - Priority 308
 - Redelivered 308
- Message Consumer Object Methods
 - Close 311
- Message Object Methods
 - Acknowledge 308
 - ClearBody 309
 - ClearProperties 309
 - GetProperty 309
 - PropertyExists 309
 - SetProperty 309
- Message Object Properties
 - CorrelationID 310
 - CorrelationIDAsBytes 310
 - DeliveryMode 310
 - Destination 310
 - Expiration 310
 - MessageID 311
 - Priority 311
 - Redelivered 311
 - ReplyTo 311
 - Timestamp 311
 - Type 311
- MessageConsumer Object Method
 - ReceiveNoWait 312
- MessageConsumer Object Methods
 - ReceiveMessage 312
- MessageConsumer Object Properties
 - MessageListener 312
 - MessageSelector 312
- MessageProducer Object Properties
 - DeliveryMode 313
 - DisableMessageID 313
 - DisableMessageTimes 313
- Queue Object Methods
 - ToString 314
- Queue Object Properties
 - QueueName 314
- QueueRequestor Object Methods
 - Create 317
 - Request 199, 318
- Session Object Methods
 - Commit 322
 - CreateBytesMessage 322
 - CreateMapMessage 322, 341
 - CreateMessage 320, 323, 347
 - CreateTextMessage 323
 - Recover 323
 - Rollback 323
 - Run 323
- Session Object Properties
 - Transacted 324
- SteamMessage Object Methods
 - Reset 326
- StreamMessage Object Methods
 - Acknowledge 324
 - ClearBody 324
 - ClearProperties 324
 - ReadBoolean 325
 - ReadByte 325
 - ReadBytes 325
 - ReadChar 325
 - ReadDouble 325
 - ReadFloat 325
 - ReadInt 326
 - ReadLong 326
 - ReadObject 326
 - ReadShort 326
 - ReadString 326
 - SetProperty 326
 - WriteBoolean 326
 - WriteBytes 327
 - WriteChar 327
 - WriteDouble 327
 - WriteFloat 328
 - WriteInt 328
 - WriteLong 328
 - WriteObject 328
 - WriteShort 328
 - WriteString 328
- StreamMessage Object Properties
 - CorrelationID 329
 - CorrelationIDAsBytes 329
 - Destination 329
 - Expiration 330
 - Priority 330
 - Redelivered 330
 - ReplyTo 330
 - Timestamp 330
 - Type 331
- TemporaryTopic Object Methods
 - Delete 331
 - ToString 331, 332
- TemporaryTopic Object Properties
 - TopicName 332
- TextMessage Object Methods
 - Acknowledge 332
 - ClearBody 332
 - ClearProperties 332
 - GetProperty 332
 - PropertyExists 333
 - SetProperty 333
- TextMessage Object Properties

- CorrelationID 333
 - CorrelationIDAsBytes 333
 - DeliveryMode 333
 - Destination 334
 - Expiration 334
 - Redelivered 335
 - ReplyTo 335
 - Timestamp 335
 - Type 335
 - Topic Object Methods
 - ToString 336
 - Topic Object Properties
 - TopicName 336
 - TopicConnection Object Methods
 - CreateTopicSession 336
 - Start 336
 - Stop 337
 - TopicConnection Object Properties
 - ClientID 337
 - MetaData 337
 - TopicConnectionFactory Object Methods
 - CreateTopic 342
 - CreateTopicConnection 337
 - TopicConnectionFactory Object Properties
 - HostName 337, 352
 - Port 337, 352
 - PortOffset 338
 - TopicPublisher Object Methods
 - Publish 338
 - TopicPublisher Object Properties
 - DeliveryMode 338
 - DisableMessageID 339
 - DisableMessageTimestamp 339
 - Priority 339
 - TimeToLive 339
 - Topic 340
 - TopicRequestor Object Methods
 - Close 198
 - Create 340
 - Request 198, 340
 - STCQueueRequestor 199
 - STCTopicRequestor 198
 - TopicSession Object Method
 - CreateTemporaryTopic 342
 - TopicSession Object Methods
 - Commit 340
 - CreateBytesMessage 341
 - CreateDurableSubscriber 341
 - CreateMessage 341
 - CreatePublisher 341
 - CreateStreamMessage 342
 - CreateSubscriber 342
 - CreateTextMessage 342, 353
 - Unsubscribe 343
 - TopicSession Object Properties
 - MessageListener 343
 - Transacted 343
 - TopicSubscriber Object Methods
 - Close 344
 - Receive 344
 - ReceiveNoWait 344
 - TopicSubscriber Object Properties
 - MessageListener 344
 - MessageSelector 344
 - NoLocal 344
 - Topic 344
 - XASession Object Methods
 - Commit 349
 - CreateBytesMessage 349
 - CreateMapMessage 349
 - CreateMessage 349
 - CreateStreamMessage 349
 - CreateTextMessage 349
 - Recover 350
 - Rollback 350
 - Run 350
 - XASession Object Properties
 - Transacted 350
 - XATopicConnection Object Methods
 - CreateTopicSession 350, 504
 - Start 351
 - Stop 351
 - XATopicConnection Object Properties
 - ClientID 351
 - MetaData 351
 - XATopicConnectionFactory Object Methods
 - CreateTopicConnection 352
 - CreateXATopicConnection 352
 - XATopicConnectionFactory Object Properties
 - PortOffset 352
 - XATopicSession Object Methods
 - Commit 353
 - CreateBytesMessage 353
 - CreateMapMessage 353
 - CreateMessage 353
 - CreateStreamMessage 353
 - Recover 353
 - Rollback 354
 - XATopicSession Object Properties
 - MessageListener 354
 - TopicSession 354
 - Transacted 354
 - JNDI 95
- L**
- LastErrorCode
 - ActiveX API for MUX 571

LastErrorText
ActiveX API for MUX 571

M

MapMessage C++
getBoolean 490
getByte 491
getBytes 491
getChar 492
getDouble 492
getfloat 493
getInt 493
getlong 493
MapMessage Object 300
MapNames Property 307
maximum client threads per e*Way 551
Message Object 308
MessageConsumer Object 311
MessageID Property 298, 307, 311, 330, 334
MessageListener Object 312
MessageListener Property 312, 323, 343, 344, 350, 354
MessageProducer Object 312
MessageSelector Property 312, 344
MetaData Property 337, 351
Multiplexer_Close subroutine 591
Multiplexer_Free subroutine 592
Multiplexer_Init subroutine 592
Multiplexer_Open subroutine 593
Multiplexer_send subroutine 594
Multiplexer_ToString subroutine 594
Multiplexer_Wait subroutine 595
MUX Instance ID 549
MUX Recovery ID 549

N

NoLocal Property 344
Non_Persistent
C++ function for JMS 509

O

OnException
C++ function for JMS 508
OnMessage 312
Open
COBOL API for MUX 565
OS/390 33

P

Persistent
C++ function for JMS 509
Port Property 300, 337, 352
PortOffset Property 338, 352
prepare
C++ function for JMS 536
Priority Property 298, 308, 311, 330, 334, 339
PropertyExists 303, 495
C function for JMS 364
propertyExists
C++ function for JMS 460, 467
PropertyExists Method 293, 309, 324, 333
publish
C++ function for JMS 521, 522, 523, 524, 525
Publish Method 338
Push IP Port 548

Q

Queue Object 314
QueueBrowser Object 314
QueueConnection Object 314
QueueConnectionFactory Object 315
QueueName Property 314
QueueReceiver Object 316
QueueReceiverClose
C function for JMS 415
QueueReceiverGetMessageSelector
C function for JMS 416
QueueReceiverGetQueue
C function for JMS 417
QueueReceiverReceive
C function for JMS 416
QueueReceiverReceiveNoWait
C function for JMS 417
QueueReceiverReceiveTimeout
C function for JMS 417
QueueRequestor Object 317
QueueRequestorClose
C function for JMS 445
QueueRequestorRequest
C function for JMS 445
QueueRequestorRequestTimeout
C function for JMS 446
QueueSender Object 318
QueueSenderClose
C function for JMS 422
QueueSenderGetDeliveryMode
C function for JMS 423, 513
QueueSenderGetDisableMessageID
C function for JMS 423
QueueSenderGetDisableMessageTimestamp

- C function for JMS 424
 - QueueSenderGetJMS_ProducerID
 - C function for JMS 424
 - QueueSenderGetPriority
 - C function for JMS 424
 - QueueSenderGetQueue
 - C function for JMS 425
 - QueueSenderGetTimeToLive
 - C function for JMS 425
 - QueueSenderSend
 - C function for JMS 426
 - QueueSenderSendEx
 - C function for JMS 426
 - QueueSenderSendToQueue
 - C function for JMS 427
 - QueueSenderSendToQueueEx
 - C function for JMS 428
 - QueueSenderSetDeliveryMode
 - C function for JMS 429
 - QueueSenderSetDisableMessageID
 - C function for JMS 429
 - QueueSenderSetDisableMessageTimestamp
 - C function for JMS 430
 - QueueSenderSetJMS_ProducerID
 - C function for JMS 430
 - QueueSenderSetPriority
 - C function for JMS 431
 - QueueSenderSetTimeToLive
 - C function for JMS 431
 - QueueSession Object 319
 - QueueSessionClose
 - C function for JMS 401
- R**
- ReadBoolean
 - C function for JMS 383
 - readBoolean
 - C++ function for JMS 483
 - ReadBoolean Method 293, 325
 - ReadByte
 - C function for JMS 383
 - readByte
 - C++ function for JMS 483
 - ReadByte Method 293, 325
 - ReadBytes
 - C function for JMS 384, 456
 - ReadBytes Message 293, 325
 - ReadChar
 - C function for JMS 384
 - readChar
 - C++ function for JMS 484
 - ReadChar Method 293, 325
 - ReadDouble
 - C function for JMS 385
 - readDouble
 - C++ function for JMS 484
 - ReadDouble Method 293, 325
 - ReadFloat
 - C function for JMS 385
 - readFloat
 - C++ function for JMS 484
 - ReadFloat Method 293, 325
 - ReadInt
 - C function for JMS 386
 - readInt
 - C++ function for JMS 484
 - ReadInt Method 294, 326
 - ReadLong
 - C function for JMS 386
 - readLong
 - C++ function for JMS 485
 - ReadLong Method 294, 326
 - ReadObject Method 326
 - ReadShort
 - C function for JMS 386
 - readShort
 - C++ function for JMS 485
 - ReadShort Method 294, 326
 - ReadString Method 326
 - ReadUnsignedByte
 - C function for JMS 387
 - readUnsignedByte
 - C++ function for JMS 485
 - ReadUnsignedByte Method 294
 - ReadUnsignedShort
 - C function for JMS 387
 - readUnsignedShort
 - C++ function for JMS 485
 - ReadUnsignedShort Method 294
 - ReadUTF
 - C function for JMS 388
 - readUTF
 - C++ function for JMS 486
 - ReadUTF Method 294
 - Receive
 - COBOL API for MUX 567
 - Receive Message Method 312
 - Receive Method 344
 - ReceiveNoWait Method 312, 344
 - recover
 - C++ function for JMS 502
 - Recover Method 323, 350, 353
 - Recovery ID, MUX 549
 - Redelivered Property 298, 308, 311, 330, 335
 - ReplyMessageAsArray
 - ActiveX API for MUX 572
 - ReplyMessageAsString

- ActiveX API for MUX 572
- ReplyMessageSize
 - ActiveX API for MUX 572
- ReplyTo Property 298, 308, 311, 330, 335
- Request 198, 199, 318, 340
- Request Reply IP Port 548
- request/reply
 - header, in Collaboration Rules 554
- request/reply header
 - header for request/reply Events 551
- Reset
 - C function for JMS 388
- reset
 - C++ function for JMS 486
- Reset Method 294, 326
- resizeMUXPool
 - Java method for MUXPooler 588
- rollback
 - C++ function for JMS 503, 534
- rollback (transaction operation)
 - defined 359
- Rollback if no Clients on Push Port 548
- Rollback Method 323, 350, 354
- Run Method 323, 350

S

- sample code for using JMS
 - compensating resource manager (CRM) 112
 - in C or RPG 125
 - in Java and COM+ 76
 - message selector 102, 104, 105, 140
 - (discussed) 101
 - publish/subscribe 78, 79, 81, 82, 126
 - (diagrammed) 78
 - queue send/receive 84, 85, 87, 131
 - (diagrammed) 84
 - request-reply 89, 90, 91, 94, 136
 - (diagrammed) 89
 - XA 106, 108, 111
- sample schema for using JMS 119
- Send
 - ActiveX API for MUX 573
 - COBOL API for MUX 566
- send
 - C++ function for JMS 517, 518, 519, 520
- send data to e*Gate, Perl subroutine 594
- Send Empty MSG When External Disconnect 548
- sendBytes
 - Java method for MUXPooler 588
- sendMessage
 - Java method for MUX 579
 - Java method for MUXPooler 588
- session modes

- AutoAcknowledge 359
- ClientAcknowledge 359
- DupsOKAcknowledge 359
- Session Object 322
- SessionCommit
 - C function for JMS 401
- SessionCreateBytesMessage
 - C function for JMS 403
- SessionCreateDurableSubscriber
 - C function for JMS 407
- SessionCreateDurableSubscriberMessageSelector
 - C function for JMS 408
- SessionCreatePublisher
 - C function for JMS 408
- SessionCreateQueue
 - C function for JMS 404
- SessionCreateReceiveMessageSelector
 - C function for JMS 405
- SessionCreateReceiver
 - C function for JMS 405
- SessionCreateSender
 - C function for JMS 406
- SessionCreateSubscriber
 - C function for JMS 409
- SessionCreateSubscriberMessageSelector
 - C function for JMS 409
- SessionCreateTemporary
 - C function for JMS 406
- SessionCreateTextMessage
 - C function for JMS 403
- SessionCreateTextMessageEx
 - C function for JMS 404
- SessionGetTransacted
 - C function for JMS 402
- SessionRecover
 - C function for JMS 402
- SessionRollback
 - C function for JMS 402
- SessionUnsubscribe
 - C function for JMS 411
- SetBoolean Method 304, 495
- SetBooleanProperty
 - C function for JMS 368
- setBooleanProperty
 - C++ function for JMS 464, 471
- SetBooleanProperty Method 294
- SetByte 304, 495
- SetByteProperty
 - C function for JMS 369
- setByteProperty
 - C++ function for JMS 464, 472
- SetBytes Method 304, 495
- SetChar Method 304, 496
- setClientID

- C++ function for JMS 500, 506
- setConnectionCount
 - Java method for MUXPooler 589
- setDebug
 - Java method for MUX 580
- setDeliveryMode
 - C++ function for JMS 514
- SetDestinationName
 - C function for JMS 414
- setDisableMessageID
 - C++ function for JMS 515
- setDisableMessageTimestamp
 - C++ function for JMS 515
- SetDouble Method 305, 496
- SetDoubleProperty
 - C function for JMS 369
- setDoubleProperty
 - C++ function for JMS 465, 472
- SetFloat Methods 305, 496
- SetFloatProperty
 - C function for JMS 370
- setFloatProperty
 - C++ function for JMS 465, 473
- setHost
 - Java constructor method for MUXPooler 589
 - Java method for MUX 581
- SetInt Method 305, 496
- SetIntProperty
 - C function for JMS 370
- setIntProperty
 - C++ function for JMS 465, 473
- SetJMS_ProducerID
 - C function for JMS 456
- setJMS_ProducerID
 - C++ function for JMS 516
- SetJMSCorrelationID
 - C function for JMS 377
- setJMSCorrelationID
 - C++ function for JMS 478
- SetJMSCorrelationIDAsBytes
 - C function for JMS 377
- setJMSCorrelationIDAsBytes
 - C++ function for JMS 478, 481
- SetJMSDeliveryMode
 - C function for JMS 378
- setJMSDeliveryMode
 - C++ function for JMS 478
- SetJMSExpiration
 - C function for JMS 378
- setJMSExpiration
 - C++ function for JMS 479
- SetJMSMessageID
 - C function for JMS 378
- setJMSMessageID
 - C++ function for JMS 479, 482
- SetJMSPriority
 - C function for JMS 379
- setJMSPriority
 - C++ function for JMS 480
- SetJMSRedelivered
 - C function for JMS 379
- setJMSRedelivered
 - C++ function for JMS 480
- SetJMSReplyTo
 - C function for JMS 380
- setJMSReplyTo
 - C++ function for JMS 480
- SetJMSTimestamp
 - C function for JMS 380
- setJMSTimestamp
 - C++ function for JMS 481
- SetJMSType
 - C function for JMS 381
- setJMSType
 - C++ function for JMS 481, 482
- SetLong Method 305, 497
- SetLongProperty
 - C function for JMS 371
- setLongProperty
 - C++ function for JMS 466, 473, 474
- SetObject Method 305, 497
- setObjectProperty
 - C++ function for JMS 466
- setPort
 - Java constructor method for MUXPooler 590
 - Java method for MUX 581
- setPriority
 - C++ function for JMS 516
- SetProperty Method 306, 309, 326, 333, 497
- setSecondsToExpire
 - Java constructor method for MUXPooler 590
 - Java method for MUX 582
- SetShort Method 306, 497
- SetShortProperty
 - C function for JMS 371
- setShortProperty
 - C++ function for JMS 467, 474
- setSleepDuration
 - Java method for MUX 582
- SetString Method 306, 497
- SetStringProperty
 - C function for JMS 372
- setStringProperty
 - C++ function for JMS 467, 475
- SetText
 - C function for JMS 395
- setText
 - C++ function for JMS 498, 499

- setTimeout
 - Java constructor method for MUXPooler 590
 - Java method for MUX 583
 - setTimeToLive
 - C++ function for JMS 516
 - setTransactionTimeout
 - C++ function for JMS 535
 - start
 - C++ function for JMS 500, 536
 - Start Method 299, 336, 351
 - stc_msclient.dll 355
 - stop
 - C++ function for JMS 500
 - Stop Method 299, 337, 351
 - StreamMessage Object 324
 - subroutines
 - Multiplexer_Close 591
 - Multiplexer_Free 592
 - Multiplexer_Init 592
 - Multiplexer_Open 593
 - Multiplexer_Send 594
 - Multiplexer_ToString 594
 - Multiplexer_Wait 595
 - Supporting Documents 36
- T**
- TemporaryQueue Object 331
 - TemporaryTopic 331
 - Text Property 335
 - TextMessage 332
 - The BytesMessage Interface for JMS in C++
 - readBoolean 483
 - readByte 483
 - readChar 484
 - readFloat 484
 - readInt 484
 - readLong 485
 - readShort 485
 - readUnsignedByte 485
 - readUnsignedShort 485
 - reset 486
 - writeBoolean 486
 - writeByte 487
 - writeBytes 487
 - writeChar 488
 - writeDouble 488
 - writeFloat 489
 - writeInt 489
 - writeLong 489
 - writeShort 490
 - The BytesMessage Interface in JMS for C++
 - readUTF 486
 - The Connectin Interface for JMS in C++
 - ConnectionStart 500
 - The Connection Interface for JMS in C++
 - close 499
 - getClientID 500
 - setClientID 500
 - stop 500
 - The createTopicSession Interface for JMS in C++
 - close 504
 - The DeliveryMode Interface for JMS in C++
 - Non_Persistent 509
 - Persistent 509
 - The ExceptionListener Interface for JMS in C++
 - OnException 508
 - The Message Interface for C++
 - setLongProperty 474
 - The Message Interface for JMS in C++
 - acknowledge 459
 - clearBody 459
 - clearProperties 460
 - getBooleanProperty 460, 468
 - getByteProperty 461
 - getDoubleProperty 461, 469
 - getFloatProperty 461, 469
 - getIntProperty 462, 469
 - getJMSCorrelationID 475
 - getJMSCorrelationIDAsBytes 475
 - getJMSExpiration 476
 - getJMSMessageID 476
 - getJMSPriority 476
 - getJMSRedelivered 477
 - getJMSReplyTo 477
 - getJMSTimestamp 477
 - getJMSType 477
 - getLongProperty 470
 - getShortProperty 463, 471
 - getStringProperty 463, 471
 - propertyExists 460, 467
 - setBooleanProperty 464, 471
 - setByteProperty 464, 472
 - setDoubleProperty 465, 472
 - setFloatProperty 465, 473
 - setIntProperty 465, 473
 - setJMSCorrelationID 478
 - setJMSCorrelationIDAsBytes 478, 481
 - setJMSDeliveryMode 478
 - setJMSExpiration 479
 - setJMSMessageID 479, 482
 - setJMSPriority 480
 - setJMSRedelivered 480
 - setJMSReplyTo 480
 - setJMSTimestamp 481
 - setJMSType 481, 482
 - setLongProperty 466, 473
 - setShortProperty 467, 474

- setStringProperty 467, 475
- The Message Interface in JMS for C++
 - getBytesProperty 468
 - getLongProperty 462
 - getPropertyName 463
- The Message Interface in JSM for C++
 - getJMSDeliveryMode 475
- The MessageProducer Interface C++ functions for JMS
 - close 513
 - getDisableMessageID 513
 - getDisableMessageTimestamp 513
 - getJMS_ProducerID 514
 - getPriority 514
 - getTimeToLive 514
 - setDeliveryMode 514
 - setDisableMessageID 515
 - setDisableMessageTimestamp 515
 - setJMS_ProducerID 516
 - setPriority 516
 - setTimeToLive 516
- The Message Interface for JMS in C++
 - setObjectProperty 466
- The Queue Interface for JMS in C++
 - getQueueName 510
 - toString 510
- The QueueConnection Interface for JMS in C++
 - createQueueSession 501
- The QueueConnectionFactory Interface for JMS in C++
 - createQueueConnection 507
- The QueueSender Interface for JMS in C++
 - send 517, 518, 519, 520
- The QueueSession Interface for JMS in C++
 - createQueue 525
 - createReceiver 526
 - createSender 527
 - createTemporaryQueue 527
- The Session Interface for JMS in C++
 - bytesMessage 503
 - close 502
 - commit 502
 - createTextMessage 503, 504
 - getTransacted 502
 - recover 502
 - rollback 503
- The TemporaryQueue Interface for JMS in C++
 - Delete 510
- The TemporaryTopic Interface for JMS in C++
 - Delete 512
- The TextMessage Interface for JMS in C++
 - getText 498
 - setText 498, 499
- The Topic Interface for JMS in C++
 - getTopicName 511
 - toString 511
- The TopicConnection Interface for JMS in C++
 - close 505
 - getClientID 505
 - getExceptionListener 506
 - setClientID 506
- The TopicConnectionFactory Interface for JMS in C++
 - createTopicConnectionFactory 508
- The TopicPublisher Interface for JMS in C++
 - publish 521, 522, 523, 524, 525
- The TopicSession Interface for JMS in C++
 - createDurableSubscriber 528
 - createPublisher 529
 - createSubscriber 529, 530
 - createTemporaryTopic 530
 - createTopic 531
 - unsubscribe 531
- The XAResource Interface for JMS in C++
 - commit 533
 - end 537
 - getTransactionTimeout 535
 - isSameRM 535
 - prepare 536
 - rollback 534
 - setTransactionTimeout 535
 - start 536
 - Xid**recover 534
- The Xid Interface for JMS in C++
 - getBranchQualifier 532
 - getFormatId 532
 - getTransactionId 532
- Timestamp Property 299, 308, 311, 330, 335
- TimeToLive Property 339
- Topic Object 335
- Topic Property 340, 344
- TopicConnection Object 336
- TopicConnectionFactory Object 337
- TopicName Property 332, 336
- TopicPublisherClose
 - C function for JMS 432
- TopicPublisherGetDeliveryMode
 - C function for JMS 433
- TopicPublisherGetDisableMessageID
 - C function for JMS 433
- TopicPublisherGetDisableMessageTimestamp
 - C function for JMS 434
- TopicPublisherGetJMS_ProducerID
 - C function for JMS 434
- TopicPublisherGetPriority
 - C function for JMS 434
- TopicPublisherGetTimeToLive
 - C function for JMS 435

- TopicPublisherGetTopic
 - C function for JMS 435
 - TopicPublisherPublish
 - C function for JMS 436
 - TopicPublisherPublishEx
 - C function for JMS 436
 - TopicPublisherPublishToTopic
 - C function for JMS 437, 524
 - TopicPublisherPublishToTopicEx
 - C function for JMS 438
 - TopicPublisherSetDeliveryMode
 - C function for JMS 438
 - TopicPublisherSetDisableMessageID
 - C function for JMS 439
 - TopicPublisherSetDisableMessageTimestamp
 - C function for JMS 439
 - TopicPublisherSetJMS_ProducerID
 - C function for JMS 440
 - TopicPublisherSetPriority
 - C function for JMS 440
 - TopicPublisherSetTimeToLive
 - C function for JMS 441
 - TopicRequestor Property 340
 - TopicRequestorClose
 - C function for JMS 443
 - TopicRequestorRequest
 - C function for JMS 442
 - TopicRequestorRequestTimeout
 - C function for JMS 443
 - TopicSession Object 338, 340
 - TopicSession Property 354
 - TopicSessionCreateTemporaryTopic
 - C function for JMS 410
 - TopicSessionCreateTopic
 - C function for JMS 411
 - TopicSubscriberClose
 - C function for JMS 418
 - TopicSubscriberGetMessageSelector
 - C function for JMS 419
 - TopicSubscriberGetNoLocal
 - C function for JMS 419
 - TopicSubscriberGetTopic
 - C function for JMS 420
 - TopicSubscriberReceive
 - C function for JMS 420
 - TopicSubscriberReceiveNoWait
 - C function for JMS 421
 - TopicSubscriberReceiveTimeout
 - C function for JMS 420
 - ToString 314
 - toString
 - C++ function for JMS 510, 511
 - ToString Method 331, 332, 336
 - Transacted Property 324, 343, 350, 354
 - transacted sessions
 - defined 359
 - Type Property 299, 308, 311, 331, 335
- ## U
- unsubscribe
 - C++ function for JMS 531
 - Unsubscribe Method 343
- ## W
- Wait
 - ActiveX API for MUX 573
 - Wait For IQ Ack 548
 - WriteBoolean
 - C function for JMS 389
 - writeBoolean
 - C++ function for JMS 486
 - WriteBoolean Method 295, 326
 - WriteByte
 - C function for JMS 389
 - writeByte
 - C++ function for JMS 487
 - WriteByte Method 295, 327
 - WriteBytes
 - C function for JMS 390, 456
 - writeBytes
 - C++ function for JMS 487
 - WriteBytes Method 295, 327
 - WriteBytesEx
 - C function for JMS 390
 - WriteChar
 - C function for JMS 391
 - writeChar
 - C++ function for JMS 488
 - WriteChar Method 295, 327
 - WriteDouble
 - C function for JMS 391
 - writeDouble
 - C++ function for JMS 488
 - WriteDouble Method 296, 327
 - WriteFloat
 - C function for JMS 392
 - writeFloat
 - C++ function for JMS 489
 - WriteFloat Method 296, 328
 - WriteInt
 - C function for JMS 392
 - writeInt
 - C++ function for JMS 489
 - WriteInt Method 296, 328
 - WriteLong
 - C function for JMS 392

Index

writeLong

 C++ function for JMS 489

WriteLong Method 296, 328

WriteObject Method 296, 328

WriteShort

 C function for JMS 393

writeShort

 C++ function for JMS 490

WriteShort Method 297, 328

WriteString Method 328

WriteUTF

 C function for JMS 393

WriteUTF Method 297

WStringToChar

 C function for JMS 453

X

XATopicSession Object 352

Xid**recover

 C++ function for JMS 534