SUN SEEBEYOND

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

Release 5.1.1

Sun microsystems

Version 20060627134230

Contents

Chapter 1

Introduction

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

What's in This Chapter

1.1 About This Document

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

1.1.1 What's in This DocumentTopic

This document includes the following chapters and appendixes:

- **Chapter 2 "Installing the eGate API Kit" on page 18** describes how to install the eGate API Kit and its samples.
- **Chapter 3 "JMS and Sun SeeBeyond JMS IQ Manager Overview" on page 21** gives an overview of JMS Messaging and the Sun SeeBeyond SJM IQ Manager.
- **Chapter 4 "Developing Java CAPS JMS Applications in C" on page 29** describes how to develop C applications to access the Sun SeeBeyond JMS IQ Manager in Java CAPS Project.
- **Chapter 5 "Developing Java CAPS JMS Applications in C++" on page 148** describes how to develop C++ applications to access the Sun SeeBeyond JMS IQ Manager in Java CAPS Project.
- **Chapter 6 "Working with the C and C++ Samples" on page 245** describes how to build and run the C and C++ samples.
- **Appendix A "C Sample Code" on page 251** provides excerpts of the sample code for each type of messaging using C.

- **Appendix B "C++ Sample Code" on page 279** provides excerpts of the sample code for each type of messaging using C++.

1.1.2 Intended Audience

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

1.1.3 Text Conventions

The following conventions are observed throughout this document.

Table 1 Text Conventions

Text Convention	Used For	Examples
Bold	Names of buttons, files, icons, parameters, variables, methods, menus, and objects	▪ Click **OK**. ▪ On the **File** menu, click **Exit**. ▪ Select the **eGate.sar** file.
`Monospaced`	Command line arguments, code samples; variables are shown in ***`bold italic`***	`java -jar ***filename***.jar`
Blue bold	Hypertext links within document	See **Related Documents** on page 16
Blue underlined	Hypertext links for Web addresses (URLs) or email addresses	http://www.sun.com

1.1.4 Screenshots

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

1.2 Related Documents

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

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

1.3 Sun Microsystems, Inc. Web Site

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

http://www.sun.com

1.4 Documentation Feedback

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

CAPS_docsfeedback@sun.com

Chapter 2

Installing the eGate API Kit

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

What's in This Chapter

2.1 Supported Operating Systems

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

2.2 System Requirements

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

In addition, you need a development environment with compiler that is compatible with the selected O/S; for example, Microsoft Visual .Net 2003 for Windows (see **Table 2 on page 19** for a list of supported compilers). For UNIX, you need GNUMake.

2.3 Supported Compilers

The table below lists the compilers that you can use to compile your C/C++ applications. If you use a different compiler, be aware that some compilers are incompatible with eGate.

Table 2 Supported Compilers

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

2.4 Installing the eGate API Kit

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

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

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

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

- the software
- the documentation
- the sample Sun Java CAPS Projects and the code samples

To install Sun SeeBeyond eGate API Kit

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

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

4 In the **Administrator > Upload** page, select the following items and click **Next** after each SAR file is selected:

- **eGate_APIKit_<OS>.sar** (for example, **eGate_APIKit_SunOS.sar**).
- **eGateAPIKitDocs.sar**

When the installation is finished, the "Installation Completed" message appears.

5 In the **Downloads** page, select **API kit for <OS>** (where <OS> is the operating system you are installing on), select a location for the **.zip** file to be saved, and then extract the file.

6 To download the sample Projects and code samples, click the **Documentation** tab, click **Download Sample**, and select a location for the **.zip** file to be saved.

For information about importing and using the sample Projects, see **Chapter 6 "Working with the C and C++ Samples"**.

2.5 Post-Installation Instructions

After the eGate API Kit installation, do the following:

1 Locate the **.dll** files in the folder where you installed eGate API Kit (these are the file extracted from the **eGate API Java** file).

2 For Windows, add the path to the library files to the **PATH** environment variable.

3 For UNIX, add the path to the library files to the library path.

Chapter 3

JMS and Sun SeeBeyond JMS IQ Manager Overview

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

What's in This Chapter

3.1 About the Sun SeeBeyond JMS IQ Manager

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

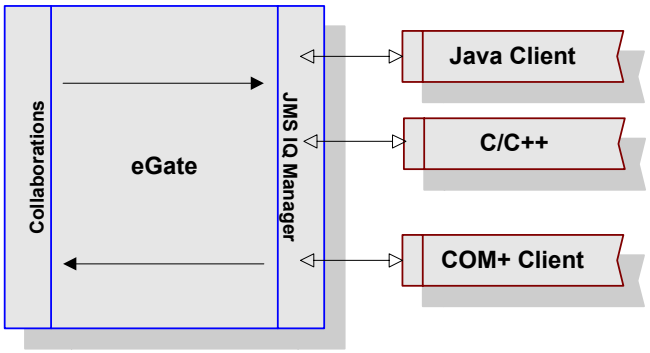
3.1.1 The Java CAPS JMS Interface

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

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

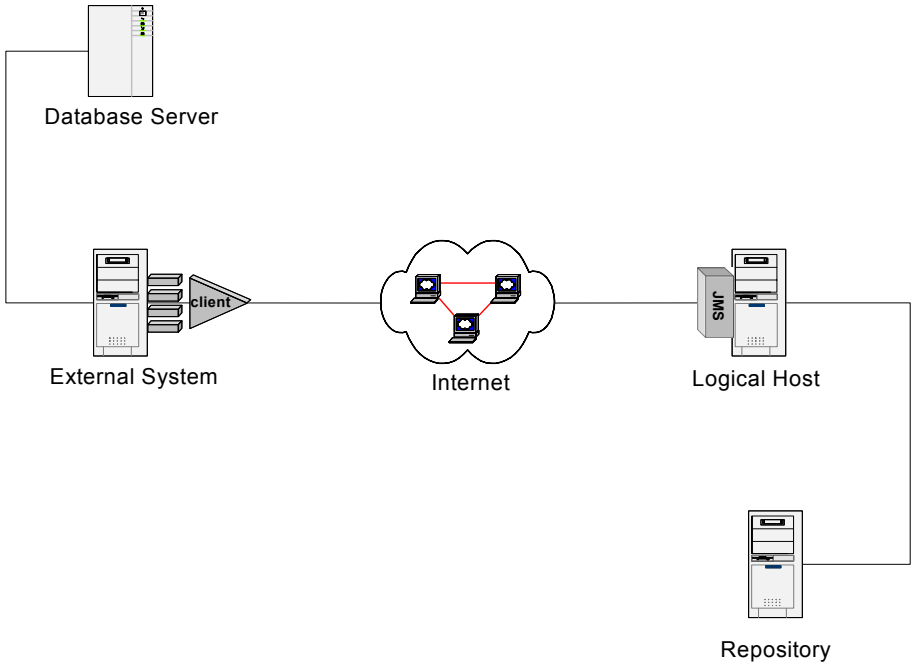
Figure 1 illustrates the communication between each component.

Figure 1 Message Service Communication Architecture



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

Figure 2 Java CAPS TCP/IP Communication Architecture



In some form, the following components must exist:

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

Important: *From this point forward, when referring to a machine, the above naming conventions are used. Remember that multiple components may reside on the same*

machine. For example, the Logical Host and the External System may exist on one physical machine.

3.1.2 Java CAPS Project Considerations

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

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

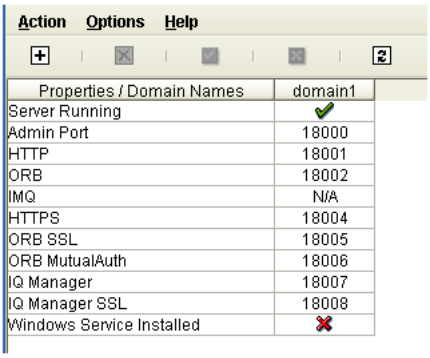
3.1.3 Viewing JMS IQ Manager Port Numbers

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

To view JMS IQ Manager port numbers

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

Figure 3 Viewing Runtime JMS IQ Manager Port Numbers



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

3.1.4 Sample Code

The eGate API Toolkit provides a sample download that includes code samples for creating interfaces to the Java CAPS JMS with C and C++. For information about implementing the sample code, see **"Working with the C and C++ Samples" on page 245**. To view the sample code, see **"C Sample Code" on page 251** and **"C++ Sample Code" on page 279**.

3.2 About the Java Messaging Service

This section provides an overview of JMS messages and some different types of messaging scenarios. The C/C++ Edition of the eGate API Kit supports the *Java Message Service Specification version 1.0.2b*.

3.2.1 JMS Messages

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

- **Header** - The header fields contain values used by both clients and providers to identify and route messages. All messages support the same set of header fields.
- **Properties** - The properties provide a way to add optional header fields to messages. They can be application-specific, standard, or provider-specific.
- **Body** (or **Payload**) - JMS supports different types of payload. The current JMS eWay Connection supports bytes and text messaging.

Message Header Fields

When a message is received by the client, the message's header is transmitted in its entirety. The fields in the header are described below.

- **JMSDestination** - The destination to which the message is being sent.
- **JMSDeliveryMode** - The mode of delivery when the message was sent. The two modes of delivery are *non-persistent* and *persistent*. Non-persistent mode causes the lowest overhead because it does not require the message to be logged to stable storage; however, non-persistent messages can be lost. Persistent mode instructs the provider to ensure that messages are not lost in transit due to provider failure.
- **JMSMessageID** - A value that uniquely identifies each message sent by a provider. The JMSMessageID is a String value that should contain a unique key for identifying messages in a historical repository. The provider must provide the scope of uniqueness. The JMSMessageID must start with the **ID:** prefix.
- **JMSTimestamp** - The specific time that a message is handed off to a provider to be sent. It is not the actual transmission time because the send may occur later due to pending transactions.
- **JMSExpiration** - A time 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, this field contains this value. If the time-to-live is specified as zero, expiration is also set to zero and the message does not expire.
- **JMSRedelivered** - An indicator of whether the message was re-delivered to the consumer. If the header is "true", the message is re-delivered; If the header is false, the message is not. The message might be marked as re-delivered if a consumer fails

to acknowledge delivery of the message, or if the JMS provider is uncertain that the consumer received the message.

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

- **JMSPriority** - The message's priority. There is a ten-level priority value system, with 0 as the lowest priority and 9 as the highest. Priorities between 0-4 are gradations of normal priority, while 5-9 are expedited priorities.
- **JMSReplyTo** - The javax.jms.Destination, which indicates the address to which to reply and enables the consumer to reply to a message associated with a specific producer.

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

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

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

Message Properties

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

Message Body (Payload)

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

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

3.2.2 JMS Messaging Types

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

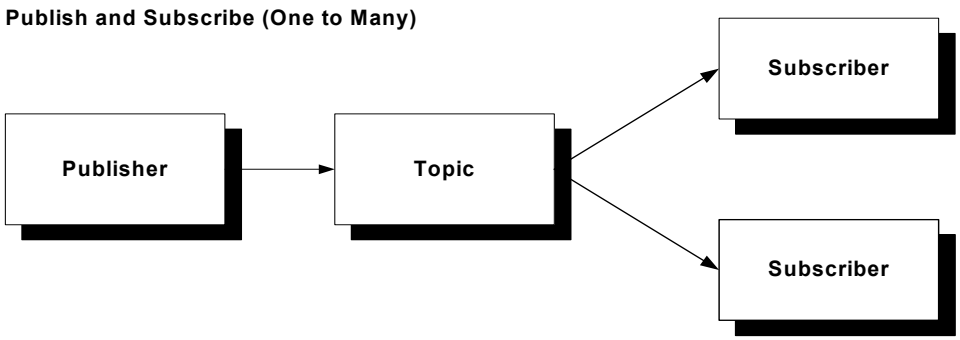
- **Publish/Subscribe Projects** on page 26
- **Point-to-Point Projects** on page 26
- **Request-Reply Projects** on page 27

Publish/Subscribe Projects

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

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

Figure 4 The Publish/Subscribe Schema



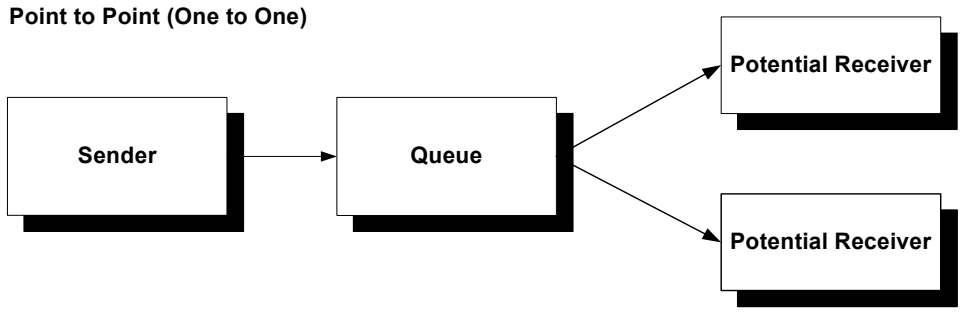
Point-to-Point Projects

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

- Message exchange takes place via a queue instead of a topic. The queue acts as a destination to which producers send messages and a source from which receivers consume messages.
- Each message is delivered to only one receiver. Multiple receivers may connect to a queue, but each message in the queue may only be consumed by one of the queue's receivers.

- The queue delivers messages to consumers in the order that they were placed in the queue by the Message Service. As messages are consumed, they are removed form the "front of the line".
- Receivers and senders can be added dynamically at runtime, allowing the system to grow as needed.

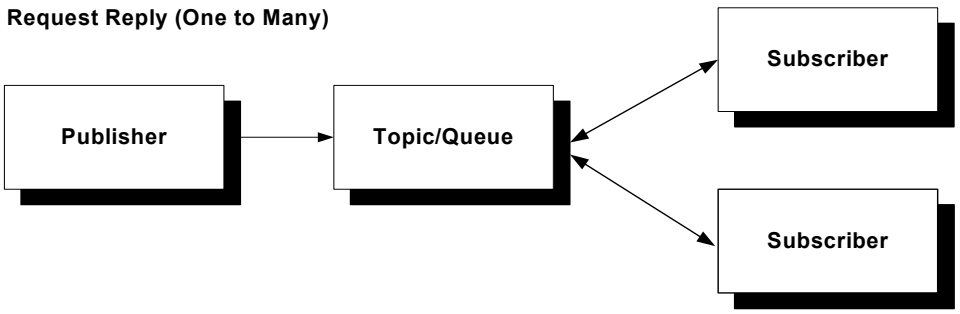
Figure 5 Point to Point



Request-Reply Projects

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

Figure 6 The Request-Reply Schema



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

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

4 The reply subscriber receives the message.

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

Chapter 4

Developing Java CAPS JMS Applications in C

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

What's in This Chapter

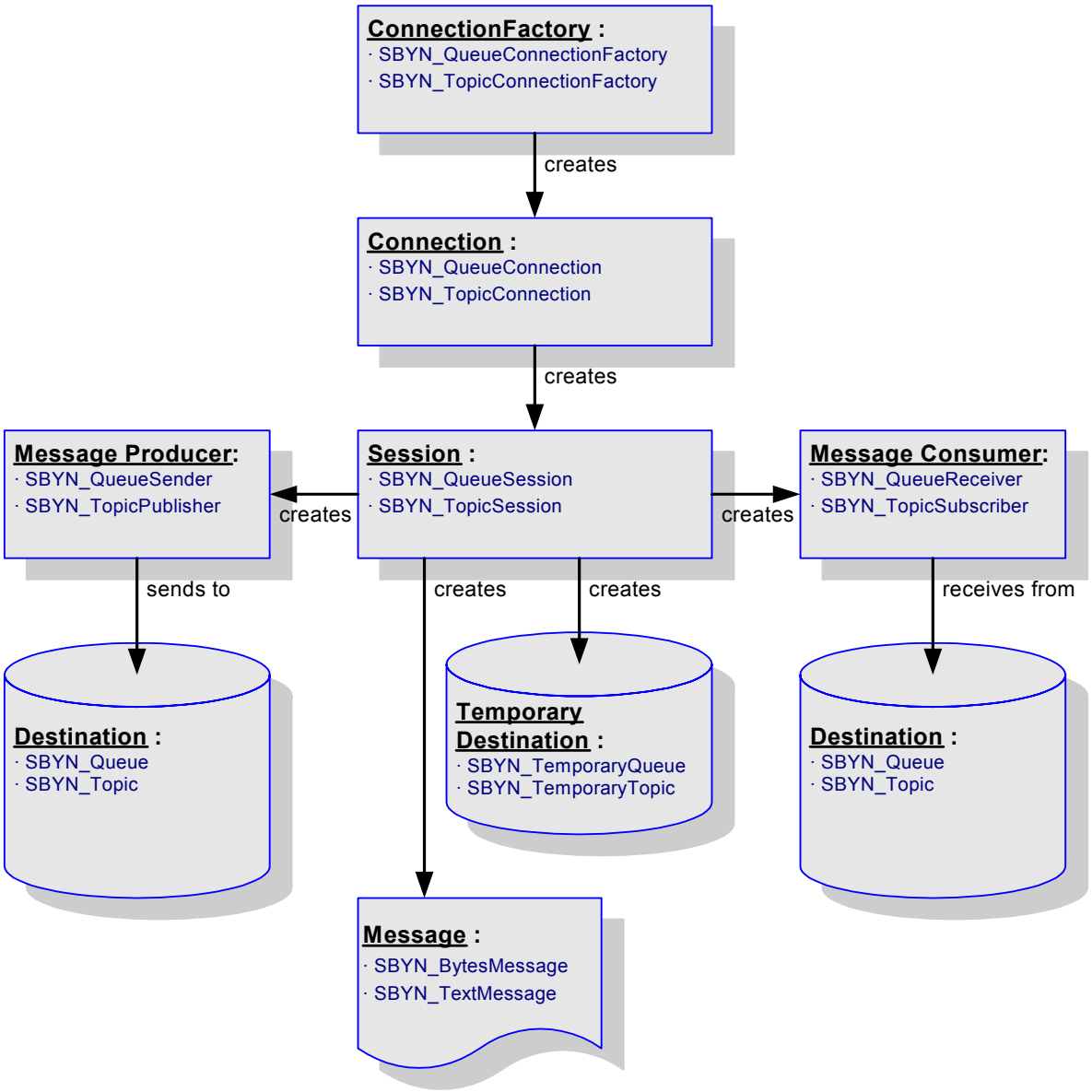
4.1 About the Java CAPS JMS Interface for C Applications

This section provides information about the C API for the JMS interface, including descriptions of the object model and wrapper functions and information about the mapping to the JMS API. The full C API reference begins with **Message Interface** on page 36.

4.1.1 The C Object Model

The eGate API Kit provides the **stc_msclient.dll** as its C interface to access the Java CAPS JMS from C applications. This section provides information about the Java CAPS JMS C Object Model (COM). **Figure 7 on page 31** shows the Java CAPS JMS C object model.

Figure 7 **C** Object Model



4.1.2 Wrapper Functions for C Applications

The API Kit supplies a set of JMS wrapper functions for C applications to allow your applications to quickly access the Java CAPS JMS without your having to understand all JMS details. While the wrapper functions are sufficient for most applications, they do not provide a complete function set; for details on the complete C API, refer to the object descriptions in this chapter.

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

- **Session**—Characterized by a hostname, port number, session type (either a pub/sub "topic session" or a point-to-point "queue session"), and connection

parameters such as acknowledgment mode, transaction mode, delivery mode, and client ID.

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

For each of these structures, the wrapper provides the equivalent of a constructor and a destructor. The other wrapper functions allow you to write, read, send, and receive messages; to set up request/reply messaging; and to commit a transacted session.

The wrapper functions use parameters *iError* and *pczError* to handle error codes and error message text.

4.1.3 C API Mapping to the JMS API

Wherever possible, the C API was designed to correspond with standard JMS Java classes. For example, the Message and BytesMessage interfaces in the C API provide similar functionality to the JMS Message and BytesMessage interfaces.

There are differences 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.

4.2 Creating Destinations

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

4.3 Using GNUMake

If you are using GNUMake on your Unix applications, select the **-k** option when you create C application executable files, to assure that if any errors are encountered, the errors are skipped and the application continues.

4.4 C Structures and Constants

This section lists and describes the C structures and constants used in the API kit.

4.4.1 C Structures

The **C** API for SeeBeyond JMS comprises the following structures:

Table 3 Structures in **C** API for SeeBeyond JMS

For Point-to-Point (P2P) Messaging	For Publish/Subscribe (Pub/Sub) Messaging
struct SBYN_QueueConnectionFactory	struct SBYN_TopicConnectionFactory
struct SBYN_QueueConnection	struct SBYN_TopicConnection
struct SBYN_QueueSession	struct SBYN_TopicSession
struct SBYN_QueueSender	struct SBYN_TopicPublisher
struct SBYN_QueueReceiver	struct SBYN_TopicSubscriber
struct SBYN_Queue	struct SBYN_Topic
struct SBYN_TemporaryQueue	struct SBYN_TemporaryTopic
struct SBYN_QueueRequestor	struct SBYN_TopicRequestor
For P2P and Pub/Sub Messaging	
struct SBYN_Destination	
struct SBYN_Message	
struct SBYN_BytesMessage	
struct SBYN_TextMessage	
struct SBYN_ConnectionMetaData	
struct SBYN_WString	
struct SBYN_WStringList	

4.4.2 C Constants

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

- **DeliveryMode Constants** on page 34
- **DestinationType Constants** on page 34

DeliveryMode Constants

Table 4 Values for DeliveryMode Constants

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

DestinationType Constants

Table 5 DestinationType Constants

Name	Value
SBYN_DESTINATION_TYPE_QUEUE	0
SBYN_DESTINATION_TYPE_TEMPORARYQUEUE	1
SBYN_DESTINATION_TYPE_TOPIC	2
SBYN_DESTINATION_TYPE_TEMPORARYTOPIC	3

MessageType Constants

Table 6 MessageType Constants

Name	Value
SBYN_MESSAGE_TYPE_MESSAGE	0
SBYN_MESSAGE_TYPE_TEXT	1
SBYN_MESSAGE_TYPE_BYTES	2

Session Constants

Table 7 Session Constants

Name	Value	Description
SBYN_AUTO_ACKNOWLEDGE	1	**1** indicates auto-acknowledgment. The session automatically acknowledges a client's receipt of a message either upon its successful return from a call to **receive** or upon successful return of the MessageListener it has called to process the message.
SBYN_CLIENT_ACKNOWLEDGE	2	**2** indicates acknowledgment by client: A client acknowledges a message by calling the message's **acknowledge** method. Acknowledging a consumed message automatically acknowledges the receipt of all messages that have been delivered by its session.
SBYN_DUPS_OK_ACKNOWLEDGE	3	**3** indicates that duplicates are acceptable, and instructs the session to lazily acknowledge message delivery This setting is likely to cause delivery of some duplicate messages if JMS fails, and should only be used by consumers that are tolerant of duplicate messages. The benefit is the reduction of session overhead, achieved by minimizing the work done to prevent duplicates.

Transacted Constants

Table 8 Transacted Constants

Name	Value
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.

Message Default Constants

Table 9 Message Default Constants

Name	Value	
SBYN_DEFAULT_DELIVERY_MODE	1	See **"DeliveryMode Constants" on page 34**.
SBYN_DEFAULT_PRIORITY	4	JMS defines a ten-level priority value: **0** is lowest priority (least expedited) and **9** is highest. Clients should consider priorities **0** through **4** as gradations of normal priority and priorities **5** through **9** as gradations of expedited priority.
SBYN_DEFAULT_TIME_TO_LIVE	0	Length of time that a produced message should be retained by the message system. Measured in milliseconds elapsed since its dispatch time. The default, **0**, has the special meaning of "retain forever" — that is, the message never expires on its own.

Miscellaneous Constants

Table 10 Miscellaneous Constants

Name	Value
DEFAULT_PORT	18007
DEFAULT_SERVER_NAME	"localhost"
MSCLIENT_DLL_NAME	"stc_msclient.dll"
FALSE	0
TRUE	1

4.5 Message Interface

The **Message** interface defines methods for working with a **Message** object. The interface includes the following methods:

Acknowledge

Syntax

```
Acknowledge(pMsg, iError, pczError)
```

Description

Acknowledges the receipt of current and previous messages.

Parameters

Name	Type	Description
pMsg	SBYN_Message*	Pointer to the message.
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.

Name	Type	Description
pczError	char*	Pointer to the error message text.

Return Value

SBYN_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

SBYN_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	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.

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

SBYN_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 34**.

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

SBYN_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.
pczError	char*	Pointer to the error message text.

Return Value

int
The priority level.

Also see **"Message Default Constants" on page 36**.

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.

Name	Type	Description
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 34**.

Parameters

Name	Type	Description
pMsg	SBYN_Message*	Pointer to the message.
iValue	int	Value corresponding to the delivery mode.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

None.

SetJMSExpiration

Syntax

```
SetJMSExpiration(pMsg, lValue, iError, pczError)
```

Description

Sets the timestamp at which the specified message is due to expire.

Parameters

Name	Type	Description
pMsg	SBYN_Message*	Pointer to the message.
lValue	long	Timestamp of the expiration.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

None.

SetJMSMessageID

Syntax

```
SetJMSMessageID(pMsg, pczValue, iError, pczError)
```

Description

Sets the message ID of the specified message.

Parameters

Name	Type	Description
pMsg	SBYN_Message*	Pointer to the message.
pczValue	char*	Pointer to the text string containing the message ID.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

None.

SetJMSPriority

Syntax

```
SetJMSPriority(pMsg, iValue, iError, pczError)
```

Description

Sets the priority level for the specified message.

Parameters

Name	Type	Description
pMsg	SBYN_Message*	Pointer to the message.
iValue	int	The priority level.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

None.

SetJMSRedelivered

Syntax

```
SetJMSRedelivered(pMsg, fValue, iError, pczError)
```

Description

Determines whether to flag the specified message as being redelivered. Used, for example, to specify redelivery for a message that has been sent but not acknowledged.

Parameters

Name	Type	Description
pMsg	SBYN_Message*	Pointer to the message.
fValue	SBYN_BOOL	Flag: If **true**, the message is being redelivered.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

None.

SetJMSReplyTo

Syntax

```
SetJMSReplyTo(pMsg, pDest, iError, pczError)
```

Description

Sets the **Destination** object where a reply to the specified message should be sent.

Parameters

Name	Type	Description
pMsg	SBYN_Message*	Pointer to the message.
pDest	SBYN_Destination*	Pointer to the **Destination** object.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

None.

SetJMSTimestamp

Syntax

```
SetJMSTimestamp(pMsg, lValue, iError, pczError)
```

Description

Sets the timestamp (JMSTimestamp header field) that the specified message was handed off to a provider to be sent. Note that this is not necessarily the time the message is actually transmitted; the actual **send** can occur later because of transactions or other client-side queueing of messages.

Parameters

Name	Type	Description
pMsg	SBYN_Message*	Pointer to the message.
lValue	long	Timestamp to be set.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

None.

SetJMSType

Syntax

```
SetJMSType(pMsg, pczJMSType, iError, pczError)
```

Description

Sets the JMSType header field, which is often used to identify the message structure and the payload type.

Parameters

Name	Type	Description
pMsg	SBYN_Message*	Pointer to the message.
pczJMSType	char*	Pointer to the text string containing the data.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

None.

4.6 Extended Message Interface

The extended message interface includes the following function:

- **GetMessageType** on page 56

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 34**.

4.7 BytesMessage Methods

A **BytesMessage** object is used for messages containing a stream of uninterpreted bytes. The receiver of the message supplies the interpretation of the bytes.

The **BytesMessage** methods include the following:

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.

Name	Type	Description
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.
pczError	char*	Pointer to the error message text.

Return Value

WriteBytesEx

Syntax

```
WriteBytesEx(pMsg, pczValue, iOffset, iLength, iError, pczError)
```

Description

Writes a portion of a byte array (unsigned char values) to the **BytesMessage** stream. For example, to extract “nag” from “manager”, set iOffset=2 and iLength=3.

Parameters

Name	Type	Description
pMsg	SBYN_Message*	Pointer to the message.
pczValue	unsigned char*	Pointer to the value to be written.
iOffset	int	The initial offset within the byte array.
iLength	int	The number of bytes to use.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

None.

WriteChar

Syntax

```
WriteChar(pMsg, cValue, iError, pczError)
```

Description

Writes an unsigned short integer to the **BytesMessage** stream.

Parameters

Name	Type	Description
pMsg	SBYN_Message*	Pointer to the message.
cValue	unsigned short	The value to be written.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

None.

WriteDouble

Syntax

```
WriteDouble(pMsg, dblValue, iError, pczError)
```

Description

Writes a double numeric value to the **BytesMessage** stream.

Parameters

Name	Type	Description
pMsg	SBYN_Message*	Pointer to the message.
dblValue	double	The value to be written.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

None.

WriteFloat

Syntax

```
WriteFloat(pMsg, fltValue, iError, pczError)
```

Description

Writes a floating-point numeric value to the **BytesMessage** stream.

Parameters

Name	Type	Description
pMsg	SBYN_Message*	Pointer to the message.
fltValue	float	The value to be written.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

None.

WriteInt

Syntax

```
WriteInt(pMsg, iValue, iError, pczError)
```

Description

Writes an integer numeric value to the **BytesMessage** stream.

Parameters

Name	Type	Description
pMsg	SBYN_Message*	Pointer to the message.
iValue	int	The value to be written.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

None.

WriteLong

Syntax

```
WriteLong(pMsg, lValue, iError, pczError)
```

Description

Writes a long numeric value to the **BytesMessage** stream.

Parameters

Name	Type	Description
pMsg	SBYN_Message*	Pointer to the message.
lValue	int	The value to be written.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

None.

WriteShort

Syntax

```
WriteShort(pMsg, nValue, iError, pczError)
```

Description

Writes a short numeric value to the **BytesMessage** stream.

Parameters

Name	Type	Description
pMsg	SBYN_Message*	Pointer to the message.
nValue	short	The value to be written.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

None.

WriteUTF

Syntax

```
WriteUTF(pMsg, pczValue, iError, pczError)
```

Description

Writes a character string to the BytesMessage stream using UTF-8 encoding.

Parameters

Name	Type	Description
pMsg	SBYN_Message*	Pointer to the message.
pczValue	char*	Pointer to the value to be written.

Name	Type	Description
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

None.

4.8 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

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.

4.9 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

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(p, iError, pczError)
```

Description

Constructs a **Connection** object; see **"Connection Interface" on page 71**.

Parameters

Name	Type	Description
p	SBYN_QueueConnectionFactory*	Pointer to the ConnectionFactory
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

SBYN_Connection* pointer.

CreateQueueConnectionEx

Syntax

```
CreateQueueConnectionEx(p, userName, password, iError, pczError)
```

Description

Constructs a **Connection** object; see **"Connection Interface" on page 71**.

Parameters

Name	Type	Description
p	SBYN_QueueConnectionFactory *	Pointer to the ConnectionFactory
userName	char *	Pointer to the user name text.
password	char *	Pointer to the password text.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

SBYN_Connection* pointer.

4.10 Connection Interface

A **Connection** object is an active connection to a JMS point-to-point provider or an active connection to a JMS pub/sub provider. A client uses a Connection to create one or more Sessions for producing and consuming messages.

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

ConnectionClose

Syntax

```
ConnectionClose(pConn, iError, pczError)
```

Description

Closes the specified connection.

Parameters

Name	Type	Description
pConnn	SBYN_Connection*	Pointer to the **Connection** object.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

None.

ConnectionGetClientID

Syntax

```
ConnectionGetClientID(pConn, iError, pczError)
```

Description

Retrieves the client ID associated with the specified **Connection** object.

Parameters

Name	Type	Description
pConn	SBYN_Connection*	Pointer to the **Connection** object.

Name	Type	Description
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

WString*

Pointer to a **WString** (wide string) object containing the text.

ConnectionSetClientID

Syntax

```
ConnectionSetClientID(pConn, pczClientID, iError, pczError)
```

Description

Sets the client ID to be associated with the specified **Connection** object. In other words, this method allows you to name the connection.

Parameters

Name	Type	Description
pConn	SBYN_Connection*	Pointer to the **Connection** object.
pczClientID	char*	Pointer to the text string for the client ID.
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 **"Session Interface" on page 74**.

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 35.
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 35.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

SBYN_Session* pointer.

4.11 Session Interface

The Session interface includes the following methods:

SessionClose

Syntax

```
SessionClose(pSessn, iError, pczError)
```

Description

Closes the specified session.

Note: *Sessions should be closed when they are no longer needed.*

Parameters

Name	Type	Description
pSessn	SBYN_Session*	Pointer to the **Session** object.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

None.

SessionCommit

Syntax

```
SessionCommit(pSessn, iError, pczError)
```

Description

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

Parameters

Name	Type	Description
pSessn	SBYN_Session*	Pointer to the **Session** object.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

None.

SessionGetTransacted

Syntax

```
SessionGetTransacted(pSessn, iError, pczError)
```

Description

Queries whether the specified session is or is not transacted.

Parameters

Name	Type	Description
pSessn	SBYN_Session*	Pointer to the **Session** object.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

SBYN_BOOL
Returns **true** if the session is transacted; otherwise, returns **false**.

SessionRecover

Syntax

```
SessionRecover(pSessn, iError, pczError)
```

Description

Stops message delivery in the specified session, causes all messages that might have been delivered but not acknowledged to be marked as **redelivered**, and restarts message delivery with the oldest unacknowledged message. Note that redelivered messages need not be delivered in the exact order they were originally delivered.

Parameters

Name	Type	Description
pSessn	SBYN_Session*	Pointer to the **Session** object.

Name	Type	Description
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

None.

SessionRollback

Syntax

```
SessionRollback(pSessn, iError, pczError)
```

Description

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

Parameters

Name	Type	Description
pSessn	SBYN_Session*	Pointer to the **Session** object.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

None.

SessionCreateBytesMessage

Syntax

```
SessionCreateBytesMessage(pSessn, iError, pczError)
```

Description

Creates a **BytesMessage**— an object used to send a message containing a stream of uninterpreted bytes.

Parameters

Name	Type	Description
pSessn	SBYN_Session*	Pointer to the **Session** object.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

SBYN_Message*
Pointer to the created message object.

SessionCreateTextMessage

Syntax

```
SessionCreateTextMessage(pSessn, iError, pczError)
```

Description

Creates an uninitialized **TextMessage**— an object used to send a message containing a string to be supplied. Also see **"SessionCreateTextMessageEx" on page 78**.

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 78**.

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 81**.

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 79**.

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_QueueSender* pointer.

SessionCreateTemporaryQueue

Syntax

```
SessionCreateTemporaryQueue(pSessn, iError, pczError)
```

Description

Creates a **Temporary** object for a specified session.

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.

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

```
SessionCreateDurableSubscriberMessageSelector(pSessn, pDest,
                        pDest, pczName, pczSelector, iError, pczError)
```

Description

Creates a durable subscriber to the specified topic, using a message selector (*pczSelector*) and/or specifying whether messages published by its own connection should be delivered to it (*fNoLocal*).

Parameters

Name	Type	Description
pSessn	SBYN_Session*	Pointer to the **Session** object.
pDest	SBYN_Destination*	Pointer to the **Destination** object.
pczName	char*	Pointer to the text string containing the client ID of the durable subscriber.
pczSelector	char*	Pointer to the string containing the text of the message selector; only messages with properties matching the message selector expression are delivered. A value of null or an empty string indicates that there is no message selector for the message consumer.
fNoLocal	SBYN_BOOL	Flag, default **false**. If **true**, inhibits the delivery of messages published by its own connection.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

SBYN_TopicSubscriber*
Pointer to the created **TopicSubscriber** object.

SessionCreatePublisher

Syntax

```
SessionCreatePublisher(pSessn, pDest, iError, pczError)
```

Description

Creates a publisher for the specified topic.

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_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

```
SessionSessionCreateTemporaryTopic(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.

4.12 TopicConnectionFactory Interface

Using pub/sub messaging, a client uses a **TopicConnectionFactory** object to create **TopicConnection** objects.

The **TopicConnectionFactory** interface includes the following methods:

CreateTopicConnectionFactory

Syntax

```
CreateTopicConnectionFactory(pczHost, iPort, iError, pczError)
```

Description

Constructs a **TopicConnectionFactory** for the specified host and port. Once constructed, it can create **TopicConnection** objects for a pub/sub JMS provider.

Parameters

Name	Type	Description
pczHost	char*	Pointer to the text of the host name.
iPort	int	Port number.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

SBYN_TopicConnectionFactory*
Pointer to the **TopicConnectionFactory** object that was created.

CreateTopicConnection

Syntax

```
CreateTopicConnection(p, iError, pczError)
```

Description

Constructs a **Connection** object; see **"Connection Interface" on page 71**.

Parameters

Name	Type	Description
p	SBYN_TopicConnectionFactory *	Pointer to the ConnectionFactory
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

SBYN_Connection* pointer.

CreateTopicConnectionEx

Syntax

```
CreateTopicConnectionEx(p, userName, password, iError, pczError);
```

Description

Constructs a **Connection** object; see **"Connection Interface" on page 71**.

Parameters

Name	Type	Description
p	SBYN_TopicConnectionFactory *	Pointer to the ConnectionFactory
userName	char *	Pointer to the user name text.
password	char *	Pointer to the password text.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

SBYN_Connection* pointer.

4.13 Destination Interface

A **Destination** object encapsulates an address for a destination provided by SeeBeyond JMS. It can also include other data, such as configuration information or metadata.

The **Destination** interface includes the following methods:

GetDestinationName

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

SBYN_WString*
Pointer to a **WString** (wide string) object containing the text.

SetDestinationName

Syntax

```
SetDestinationName(pDest, pczName, iError, pczError)
```

Description

Sets the name of the specified **Destination** object.

Parameters

Name	Type	Description
pDest	SBYN_Destination*	Pointer to the **Destination** object.

Name	Type	Description
pczName	char*	Pointer to the text string containing the name of the destination.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

None.

DestinationToString

Syntax

```
DestinationToString(pDest, iError, pczError)
```

Description

Retrieves a text string representation of the specified **Destination** object.

Parameters

Name	Type	Description
pDest	SBYN_Destination*	Pointer to the **Destination** object.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

SBYN_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.

4.14 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

Syntax

```
QueueReceiverClose(pQueRecvr, iError, pczError)
```

Description

Closes the specified queue receiver.

Note: *When a message consumer is no longer needed, it should be closed.*

Parameters

Name	Type	Description
pQueRecvr	SBYN_QueueReceiver*	Pointer to the **QueueReceiver** object.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

None.

QueueReceiverGetMessageSelector

Syntax

```
QueueReceiverGetMessageSelector(pQueRecvr, iError, pczError)
```

Description

Retrieves the message selector expression associated with this queue receiver.

Parameters

Name	Type	Description
pQueRecvr	SBYN_QueueReceiver*	Pointer to the **QueueReceiver** object.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

char*
Pointer to the string containing the message selector text. Returns null if no message selector exists, or if the message selector was set to null or the empty string.

QueueReceiverReceive

Syntax

```
QueueReceiverReceive(pQueRecvr, iError, pczError)
```

Description

Receives the next message produced for this queue. If the called within a transaction, the queue retains the message until the transaction commits.

Parameters

Name	Type	Description
pQueRecvr	SBYN_QueueReceiver*	Pointer to the **QueueReceiver** object.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

SBYN_Message*
Pointer to the received message.

QueueReceiverReceiveTimeOut

Syntax

```
QueueReceiverReceiveTimeOut(pQueRecvr, lTimeout, iError, pczError)
```

Description

Receives the next message that arrives within the specified timeout interval. A timeout value of 0 makes this function equivalent to **"QueueReceiverReceive" on page 91**.

Parameters

Name	Type	Description
pQueRecvr	SBYN_QueueReceiver*	Pointer to the **QueueReceiver** object.
lTimeout	long	The timeout value.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

SBYN_Message*
Pointer to the received message, or null if no message is available.

QueueReceiverReceiveNoWait

Syntax

```
QueueReceiverReceiveNoWait(pQueRecvr, iError, pczError)
```

Description

Receives the next message produced for this queue if one is immediately available.

Parameters

Name	Type	Description
pQueRecvr	SBYN_QueueReceiver*	Pointer to the **QueueReceiver** object.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

SBYN_Message*
Pointer to the received message, or null if no message is available.

QueueReceiverGetQueue

Syntax

```
QueueReceiverGetQueue(pQueRecvr, iError, pczError)
```

Description

Retrieves the queue associated with the specified queue receiver.

Parameters

Name	Type	Description
pQueRecvr	SBYN_QueueReceiver*	Pointer to the **QueueReceiver** object.

Name	Type	Description
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

SBYN_Destination*
Pointer to the queue.

4.15 TopicSubscriber Interface

The TopicSubscriber interface includes the following methods:

TopicSubscriberClose

Syntax

```
TopicSubscriberClose(pTopSub, iError, pczError)
```

Description

Closes the specified topic subscriber.

Note: *When a message consumer is no longer needed, it should be closed.*

Parameters

Name	Type	Description
pTopSub	SBYN_TopicSubscriber*	Pointer to the **TopicSubscriber** object.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

None.

TopicSubscriberGetMessageSelector

Syntax

```
TopicSubscriberGetMessageSelector(pTopSub, iError, pczError)
```

Description

Retrieves the message selector expression associated with the specified topic subscriber.

Parameters

Name	Type	Description
pTopSub	SBYN_TopicSubscriber*	Pointer to the **TopicSubscriber** object.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

char*

Pointer to the string containing the message selector text. Returns null if no message selector exists, or if the message selector was set to null or the empty string.

TopicSubscriberGetNoLocal

Syntax

```
TopicSubscriberGetNoLocal(pTopSub, iError, pczError)
```

Description

Queries whether the NoLocal flag is set for the specified topic subscriber.

Parameters

Name	Type	Description
pTopSub	SBYN_TopicSubscriber*	Pointer to the **TopicSubscriber** object.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

SBYN_BOOL

If **false** (the default), the subscriber can also publish via the same connection.
If **true**, delivery of messages published via its own connection is inhibited.

TopicSubscriberGetTopic

Syntax

```
TopicSubscriberGetTopic(pTopSub, iError, pczError)
```

Description

Retrieves the topic associated with the specified topic subscriber.

Parameters

Name	Type	Description
pTopSub	SBYN_TopicSubscriber*	Pointer to the **TopicSubscriber** object.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

SBYN_Destination*
Pointer to the topic.

TopicSubscriberReceive

Syntax

```
TopicSubscriberReceive(pTopSub, iError, pczError)
```

Description

Receives the next message produced for this topic. If called within a transaction, the topic retains the message until the transaction commits.

Parameters

Name	Type	Description
pTopSub	SBYN_TopicSubscriber*	Pointer to the **TopicSubscriber** object.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

SBYN_Message*
Pointer to the received message.

TopicSubscriberReceiveTimeout

Syntax

```
TopicSubscriberReceiveTimeOut(pTopSub, lTimeout, iError, pczError)
```

Description

Receives the next message that arrives within the specified timeout interval. A timeout value of 0 makes this function equivalent to **"TopicSubscriberReceive" on page 95**.

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)
```

4.16 QueueSender Interface

Using point-to-point messaging, a client uses a **QueueSender** object to send messages to a queue. After sending a message to a queue, a client may retain the message and modify it without affecting the message that has been sent. The same message object may be sent multiple times.

The **QueueSender** interface includes the following methods:

QueueSenderClose

Syntax

```
QueueSenderClose(pQueSender, iError, pczError)
```

Description

Closes the specified queue sender.

Note: *When a message producer is no longer needed, it should be closed.*

Parameters

Name	Type	Description
pQueSender	SBYN_QueueSender*	Pointer to the **QueueSender** object.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

None.

QueueSenderGetDeliveryMode

Syntax

```
QueueSenderGetDeliveryMode(pQueSender, iError, pczError)
```

Description

Retrieves the value of the **DeliveryMode** property of the specified queue sender.

Parameters

Name	Type	Description
pQueSender	SBYN_QueueSender*	Pointer to the **QueueSender** object.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

int
See **"DeliveryMode Constants" on page 34**.

QueueSenderGetDisableMessageID

Syntax

```
QueueSenderGetDisableMessageID(pQueSender, iError, pczError)
```

Description

Queries whether message IDs are or are not disabled for the specified queue sender.

Parameters

Name	Type	Description
pQueSender	SBYN_QueueSender*	Pointer to the **QueueSender** object.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

SBYN_BOOL
Returns **true** if message IDs are disabled; otherwise, returns **false**.

QueueSenderGetDisableMessageTimestamp

Syntax

```
QueueSenderGetDisableMessageTimestamp(pQueSender, iError, pczError)
```

Description

Queries whether message timestamping is or is not disabled for the specified queue sender.

Parameters

Name	Type	Description
pQueSender	SBYN_QueueSender*	Pointer to the **QueueSender** object.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

int
Returns **1** if message timestamping is disabled; otherwise, returns **0**.

QueueSenderGetJMS_ProducerID

Syntax

```
QueueSenderGetJMS_ProducerID(pQueSender, iError, pczError)
```

Description

Retrieves the value of the **ProducerID** property for the specified queue sender.

Parameters

Name	Type	Description
pQueSender	SBYN_QueueSender*	Pointer to the **QueueSender** object.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

SBYN_WString*
Pointer to a **WString** (wide string) object containing the text.

QueueSenderGetPriority

Syntax

```
QueueSenderGetPriority(pQueSender, iError, pczError)
```

Description

Queries the value of the message **Priority** property of the specified queue sender.

Parameters

Name	Type	Description
pQueSender	SBYN_QueueSender*	Pointer to the **QueueSender** object.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

int
Message priority level, from **0** (least expedited) through **9** (most expedited). See **"Message Default Constants" on page 36**.

QueueSenderGetQueue

Syntax

```
QueueSenderGetQueue(pQueSender, iError, pczError)
```

Description

Retrieves the queue associated with the queue sender.

Parameters

Name	Type	Description
pQueSender	SBYN_QueueSender*	Pointer to the **QueueSender** object.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

SBYN_Destination*
Pointer to the queue.

QueueSenderGetTimeToLive

Syntax

```
QueueSenderGetTimeToLive(pQueSender, iError, pczError)
```

Description

Queries the value of the **TimeToLive** property of the specified queue sender.

Parameters

Name	Type	Description
pQueSender	SBYN_QueueSender*	Pointer to the **QueueSender** object.

Name	Type	Description
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 **"Message Default Constants" on page 36**.

QueueSenderSend

Syntax

```
QueueSenderSend(pQueSender, pMsg, iError, pczError)
```

Description

Sends the specified message to the queue, using default values for properties (**DeliveryMode**, **Priority**, and **TimeToLive**) of the specified queue sender. If you need to override the default values, see **"QueueSenderSendEx" on page 101**.

To specify the queue destination, see **"QueueSenderSendToQueue" on page 102**.

Parameters

Name	Type	Description
pQueSender	SBYN_QueueSender*	Pointer to the **QueueSender** object.
pMsg	SBYN_Message*	Pointer to the message to send.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

None.

QueueSenderSendEx

Syntax

```
QueueSenderSendEx(pQueSender, pMsg,
                  iDelivMode, iPriority, lMilSecToLive,
                  iError, pczError)
```

Description

Sends the specified message to the queue, overriding one or more default values for properties of the specified queue sender. Compare to **"QueueSenderSend" on page 101**.

To specify the queue destination, see **"QueueSenderSendToQueueEx" on page 103**.

Parameters

Name	Type	Description
pQueSender	SBYN_QueueSender*	Pointer to the **QueueSender** object.
pMsg	SBYN_Message*	Pointer to the message to send.
iDelivMode	int	Value to be used for **DeliveryMode**; see **"DeliveryMode Constants" on page 34**.
iPriority	int	Value to be used for **Priority**; see **"Message Default Constants" on page 36**.
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 **"Message Default Constants" on page 36**.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

None.

QueueSenderSendToQueue

Syntax

```
QueueSenderSendToQueue(pQueSender, pDest, pMsg, iError, pczError)
```

Description

Sends the specified message to the specified queue, using default values for properties (**DeliveryMode**, **Priority**, and **TimeToLive**) of the specified queue sender. Compare to **"QueueSenderSendToQueueEx" on page 103**.

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 101**.

Parameters

Name	Type	Description
pQueSender	SBYN_QueueSender*	Pointer to the **QueueSender** object.
pDest	SBYN_Destination*	Pointer to the queue.
pMsg	SBYN_Message*	Pointer to the message to send.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

None.

QueueSenderSendToQueueEx

Syntax

```
QueueSenderSendToQueueEx(pQueSender, pDest, pMsg,
                         iDelivMode, iPriority, lMilSecToLive,
                         iError, pczError)
```

Description

Sends the specified message to the specified queue, overriding one or more default values for properties of the specified queue sender. Compare to **"QueueSenderSendToQueue" on page 102**.

Parameters

Name	Type	Description
pQueSender	SBYN_QueueSender*	Pointer to the **QueueSender** object.
pDest	SBYN_Destination*	Pointer to the queue.
pMsg	SBYN_Message*	Pointer to the message to send.
iDelivMode	int	Value to be used for **DeliveryMode**; see **"DeliveryMode Constants" on page 34**.
iPriority	int	Value to be used for **Priority**; see **"Message Default Constants" on page 36**.
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 **"Message Default Constants" on page 36**.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

None.

QueueSenderSetDeliveryMode

Syntax

```
QueueSenderSetDeliveryMode(pQueSender, iDelivMode, iError, pczError)
```

Description

Sets the value of the **DeliveryMode** property of the specified queue sender. See **"DeliveryMode Constants" on page 34**.

Parameters

Name	Type	Description
pQueSender	SBYN_QueueSender*	Pointer to the **QueueSender** object.
iDelivMode	int	Value for the **DeliveryMode** property.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

None.

QueueSenderSetDisableMessageID

Syntax

```
QueueSenderSetDisableMessageID(pQueSender, fDisabled,
                               iError, pczError)
```

Description

Determines whether message IDs are disabled for this queue sender. Default **false**.

Parameters

Name	Type	Description
pQueSender	SBYN_QueueSender*	Pointer to the **QueueSender** object.
fDisabled	SBYN_BOOL	Flag, default **false**; if **true**, message IDs are disabled.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

None.

QueueSenderSetDisableMessageTimestamp

Syntax

```
QueueSenderSetDisableMessageTimestamp(pQueSender, fDisabled,
                                      iError, pczError)
```

Description

Determines whether message timestamping is disabled for this queue sender. Default **false**.

Since message timestamps take effort to create and increase the size of a message, this flag can be set **true** to reduce overhead if message IDs are not used by an application.

Parameters

Name	Type	Description
pQueSender	SBYN_QueueSender*	Pointer to the **QueueSender** object.
fDisabled	SBYN_BOOL	Flag, default **false**; if **true**, message timestamping is disabled.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

None.

QueueSenderSetJMS_ProducerID

Syntax

```
QueueSenderSetJMS_ProducerID(pQueSender, pczProdID, iError, pczError)
```

Description

Sets the value of the **ProducerID** property for the specified queue sender.

Parameters

Name	Type	Description
pQueSender	SBYN_QueueSender*	Pointer to the **QueueSender** object.
pczProdID	char*	Pointer to text string containing the Producer ID.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

None.

QueueSenderSetPriority

Syntax

```
QueueSenderSetPriority(pQueSender, iPriority, iError, pczError)
```

Description

Sets the value of the message **Priority** property, from **0** (least expedited) through **9** (most expedited). See **"Message Default Constants" on page 36**.

Parameters

Name	Type	Description
pQueSender	SBYN_QueueSender*	Pointer to the **QueueSender** object.
iPriority	int	Message priority level.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

None.

QueueSenderSetTimeToLive

Syntax

```
QueueSenderSetTimeToLive(pQueSender, lMilSecToLive, iError, pczError)
```

Description

Sets the value of the **TimeToLive** property of the specified queue sender.

Parameters

Name	Type	Description
pQueSender	SBYN_QueueSender*	Pointer to the **QueueSender** object.
lMilSecToLive	long	Value to be used for TimeToLive: Length of time (milliseconds elapsed since dispatch time) that a produced message should be retained by JMS. See **"Message Default Constants" on page 36**.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

None.

4.17 TopicPublisher Interface

The TopicPublisher interface includes the following methods:

TopicPublisherClose

Syntax

```
TopicPublisherClose(pTopPub, iError, pczError)
```

Description

Closes the specified topic publisher.

Note: *When a message producer is no longer needed, it should be closed.*

Parameters

Name	Type	Description
pTopPub	SBYN_TopicPublisher*	Pointer to the **TopicPublisher** object.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

None.

TopicPublisherGetDeliveryMode

Syntax

```
TopicPublisherGetDeliveryMode(pTopPub, iError, pczError)
```

Description

Retrieves the value of the **DeliveryMode** property of the specified topic publisher.

Parameters

Name	Type	Description
pTopPub	SBYN_TopicPublisher*	Pointer to the **TopicPublisher** object.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

int

See **"DeliveryMode Constants" on page 34**.

TopicPublisherGetDisableMessageID

Syntax

```
TopicPublisherGetDisableMessageID(pTopPub, iError, pczError)
```

Description

Queries whether message IDs are or are not disabled for the specified topic publisher.

Parameters

Name	Type	Description
pTopPub	SBYN_TopicPublisher*	Pointer to the **TopicPublisher** object.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

SBYN_BOOL

Returns **true** if message IDs are disabled; otherwise, returns **false**.

TopicPublisherGetDisableMessageTimestamp

Syntax

```
TopicPublisherGetDisableMessageTimestamp(pTopPub, iError, pczError)
```

Description

Queries whether message IDs are or are not disabled for the specified topic publisher.

Parameters

Name	Type	Description
pTopPub	SBYN_TopicPublisher*	Pointer to the **TopicPublisher** object.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

int
Returns **true** if message timestamping is disabled; otherwise, returns **false**.

TopicPublisherGetJMS_ProducerID

Syntax

```
TopicPublisherGetJMS_ProducerID(pTopPub, iError, pczError)
```

Description

Retrieves the value of the **ProducerID** property for the specified topic publisher.

Parameters

Name	Type	Description
pTopPub	SBYN_TopicPublisher*	Pointer to the **TopicPublisher** object.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

SBYN_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.

Name	Type	Description
pczError	char*	Pointer to the error message text.

Return Value

int
Message priority level, from **0** (least expedited) through **9** (most expedited). See **"Message Default Constants" on page 36**.

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 **"Message Default Constants" on page 36**.

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 111**.

To specify the topic destination, see **"TopicPublisherPublishToTopic" on page 112**.

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 111**.

To specify the topic destination, see **"TopicPublisherPublishToTopicEx" on page 113**.

Parameters

Name	Type	Description
pTopPub	SBYN_TopicPublisher*	Pointer to the **TopicPublisher** object.
pMsg	SBYN_Message*	Pointer to the message to be published.

Name	Type	Description
iDelivMode	int	Value to be used for **DeliveryMode**; see **"DeliveryMode Constants" on page 34**.
iPriority	int	Value to be used for **Priority**; see **"Message Default Constants" on page 36**.
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 **"Message Default Constants" on page 36**.
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 113**.

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 111**.

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 112**.

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 34**.
iPriority	int	Value to be used for **Priority**; see **"Message Default Constants" on page 36**.
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 **"Message Default Constants" on page 36**.
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 34**.

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 **"Message Default Constants" on page 36**.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

None.

TopicPublisherSetTimeToLive

Syntax

```
TopicPublisherSetTimeToLive(pTopPub, lMSecToLive, iError, pczError)
```

Description

Sets the value of the **TimeToLive** property for the specified topic publisher.

Parameters

Name	Type	Description
pTopPub	SBYN_TopicPublisher*	Pointer to the **TopicPublisher** object.
lMilSecToLive	long	Length of time (milliseconds elapsed since dispatch time) that a produced message should be retained by JMS. See **"Message Default Constants" on page 36**.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

None.

4.18 TopicRequestor Interface

The **TopicRequestor** object is used in request/reply messaging to simplify making service requests. Given a non-transacted **TopicSession** object and a destination **Topic**, it creates a **TemporaryTopic** object for the responses and provides a request method that sends the request message and waits for its reply.

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

CreateTopicRequestor

Syntax

```
CreateTopicRequestor(pTopSessn, pDest, iError, pczError)
```

Description

Constructs a **TopicRequestor** object. This implementation assumes that the parent topic session is non-transacted and that the value of its **DeliveryMode** property is either AUTO_ACKNOWLEDGE or DUPS_OK_ACKNOWLEDGE.

Parameters

Name	Type	Description
pTopSessn	SBYN_TopicSession*	Pointer to the parent **TopicSession** object.
pDest	SBYN_Destination*	Pointer to the destination topic.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

SBYN_TopicRequestor*
Pointer to the constructed **TopicRequestor** object.

TopicRequestorRequest

Syntax

```
TopicRequestorRequest(pTopReq, pMsg, iError, pczError)
```

Description

Sends a request and waits for a reply, without any upper limit on the time to wait (compare **"TopicRequestorRequestTimeout" on page 118**). 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.

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_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 117**.

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.

4.19 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

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 121**). 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 120**.

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.

4.20 Destructor Methods

Destructor methods include the following:

DeleteQueueConnectionFactory

Syntax

```
DeleteQueueConnectionFactory(pQueCxnFac, iError, pczError)
```

Description

Deletes the specified queue connection factory.

Parameters

Name	Type	Description
pQueCxnFac	SBYN_QueueConnectionFactory*	Pointer to the **QueueConnectionFactory** object.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

None.

DeleteConnection

Syntax

```
DeleteConnection(pQueCxn, iError, pczError)
```

Description

Deletes the specified queue connection.

Parameters

Name	Type	Description
pQueCxn	SBYN_Connection*	Pointer to the **QueueConnection** object.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

None.

DeleteQueueReceiver

Syntax

```
DeleteQueueReceiver(pQueRecvr, iError, pczError)
```

Description

Deletes the specified queue receiver.

Parameters

Name	Type	Description
pQueRecvr	SBYN_QueueReceiver*	Pointer to the **QueueReceiver** object.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

None.

DeleteQueueRequestor

Syntax

```
DeleteQueueRequestor(pQueReq, iError, pczError)
```

Description

Deletes the specified queue requestor.

Parameters

Name	Type	Description
pQueReq	SBYN_QueueRequestor *	Pointer to the **QueueRequestor** object.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

None.

DeleteQueueSender

Syntax

```
DeleteQueueSender(pQueSender, iError, pczError)
```

Description

Deletes the specified queue sender.

Parameters

Name	Type	Description
pQueSender	SBYN_QueueSender*	Pointer to the **QueueSender** object.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

None.

DeleteSession

Syntax

```
DeleteSession(pQueSessn, iError, pczError)
```

Description

Deletes the specified queue session.

Parameters

Name	Type	Description
pQueSessn	SBYN_Session*	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.

DeleteConnection

Syntax

```
DeleteConnection(pTopCxn, iError, pczError)
```

Description

Deletes the specified topic connection.

Parameters

Name	Type	Description
pTopCxn	SBYN_Connection*	Pointer to the **TopicConnection** object.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

None.

DeleteSession

Syntax

```
DeleteSession(pTopSession, iError, pczError)
```

Description

Deletes the specified topic session.

Parameters

Name	Type	Description
pTopSessn	SBYN_Session*	Pointer to the **TopicSession** object.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

None.

DeleteTopicSubscriber

Syntax

```
DeleteTopicSubscriber(pTopSub, iError, pczError)
```

Description

Deletes the specified topic subscriber.

Parameters

Name	Type	Description
pTopSub	SBYN_TopicSubscriber*	Pointer to the **TopicSubscriber** object.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

None.

DeleteTopicRequestor

Syntax

```
DeleteTopicRequestor(pTopReq, iError, pczError)
```

Description

Deletes the specified topic requestor.

Parameters

Name	Type	Description
qTopReq	SBYN_TopicRequestor*	Pointer to the **TopicRequestor** object.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

None.

DeleteTopicPublisher

Syntax

```
DeleteTopicPublisher(pTopPub, iError, pczError)
```

Description

Deletes the specified topic publisher.

Parameters

Name	Type	Description
pTopPub	SBYN_TopicPublisher*	Pointer to the **TopicPublisher** object.
iError	int*	Pointer to the error number.
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.

4.21 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:

Also see **"WStringList Helper Interface" on page 129**.

CharToWString

Syntax

```
CharToWString(pczStr)
```

Description

Translates the specified character array to a **WString** (wide string) object.

Parameters

Name	Type	Description
pczStr	const char*	Pointer to the character array.

Return Value

SBYN_WString*
Pointer to the **WString** object containing the translation of the character array.

DeleteWString

Syntax

```
DeleteWString(WStr)
```

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)
```

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.

4.22 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:

Also see **"WString Helper Interface" on page 127**.

DeleteWStringList

Syntax

```
DeleteWString(WStrList)
```

Description

This method 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

This method 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.

4.23 XA Topic Methods

In XA pub/sub messaging, a client uses an **XATopicConnectionFactory** object to create Connection objects. The XA Topic includes the following methods:

CreateXATopicConnectionFactory

Syntax

```
CreateXATopicConnectionFactory(pHostname, usPort, iError, pczError)
```

Description

Constructs an **XATopicConnectionFactory** for the specified host and port. Once constructed, it can create Connection objects for an XA pub/sub JMS provider.

Parameters

Name	Type	Description
pHostname	char*	Pointer to the host name.
usPort	unsigned short	Port number.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

SBYN_XATopicConnectionFactory*
Pointer to the Connection object that was created.

CreateXATopicConnection

Syntax

```
CreateXATopicConnection(pXATcf, iError, pczError)
```

Description

Constructs a **Connection** object; see **"Connection Interface" on page 71**.

Parameters

Name	Type	Description
pXATcf	SBYN_XATopicConnectionFactory *	Pointer to the ConnectionFactory.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

SBYN_Connection* pointer.

CreateXATopicConnectionEx

Syntax

```
CreateXATopicConnectionEx(pXATcf, userName, password, iError,
pczError);
```

Description

Constructs a **Connection** object; see **"Connection Interface" on page 71**.

Parameters

Name	Type	Description
pXATcf	SBYN_XATopicConnectionFactory *	Pointer to the ConnectionFactory.
userName	char *	Pointer to the user name text.
password	char *	Pointer to the password text.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

SBYN_Connection* pointer.

XAConnectionCreateTopicSession

Syntax

```
XAConnectionCreateTopicSession(pXATc, bTransMode, iAckMode, iError,
pczError)
```

Description

Constructs a **Session** object; see **"Session Interface" on page 74**.

Parameters

Name	Type	Description
pXATc	SBYN_Connection*	Pointer to the Connection object.
bTransMode	SBYN_BOOL	Indicator of the transaction mode (see **"Transacted Constants" on page 35**).
iAckMode	int	Indicator of the acknowledgement mode (see **"Session Constants" on page 35**).
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

SBYN_Session* pointer.

XAConnectionCreateXATopicSession

Syntax

```
XAConnectionCreateXATopicSession(pXATc, iError, pczError)
```

Description

Constructs a **Session** object; see **"Session Interface" on page 74**.

Parameters

Name	Type	Description
pXATc	SBYN_Connection*	Pointer to the Connection object.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

SBYN_Session* pointer.

DeleteXATopicConnectionFactory

Syntax

```
DeleteXATopicConnectionFactory(pXATcf, iError, pczError)
```

Description

Deletes the **ConnectionFactory**.

Parameters

Name	Type	Description
pXATcf	SBYN_XATopicConnectionFactory*	Pointer to the ConnectionFactory.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

None.

4.24 XA Queue Methods

In XA point-to-point messaging, a client uses an **XAQueueConnectionFactory** object to create Connection objects. The XA Queue includes the following methods:

CreateXAQueueConnectionFactory

Syntax

```
CreateXAQueueConnectionFactory(pHostname, usPort, iError, pczError)
```

Description

Constructs a **XAQueueConnectionFactory** object for the specified host and port. Once constructed, it can create **XAQueueConnection** objects for an XA point-to-point JMS provider.

Parameters

Name	Type	Description
pHostname	char*	Pointer to the text of the host name.
usPort	unsigned short	Port number.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

SBYN_XAQueueConnectionFactory*
Pointer to the **XAQueueConnectionFactory** object that was created.

CreateXAQueueConnection

Syntax

```
CreateXAQueueConnection(pXAQcf, iError, pczError)
```

Description

Constructs a **Connection** object; see **"Connection Interface" on page 71**.

Parameters

Name	Type	Description
pXAQcf	SBYN_XAQueueConnectionFactory*	Pointer to the ConnectionFactory
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

SBYN_Connection* pointer.

CreateXAQueueConnectionEx

Syntax

```
CreateXAQueueConnectionEx(pXAQcf, userName, password, iError,
pczError)
```

Description

Constructs a **Connection** object; see **"Connection Interface" on page 71**.

Parameters

Name	Type	Description
pXAQcf	SBYN_XAQueueConnectionFactory *	Pointer to the ConnectionFactory
userName	char *	Pointer to the user name text.
password	char *	Pointer to the password text.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

SBYN_Connection* pointer.

XAConnectionCreateQueueSession

Syntax

```
XAConnectionCreateQueueSession(pXAQc, bTransMode, iAckMode, iError,
pczError)
```

Description

Constructs a **Session** object; see **"Session Interface" on page 74**.

Parameters

Name	Type	Description
pXAQc	SBYN_Connection*	Pointer to the Connection object.
bTransMode	SBYN_BOOL	Indicator of the transaction mode (see **"Transacted Constants" on page 35**).
iAckMode	int	Indicator of the acknowledgement mode (see **"Session Constants" on page 35**).
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

SBYN_Session* pointer.

XAConnectionCreateXAQueueSession

Syntax

```
XAConnectionCreateXAQueueSession(pXAQc, iError, pczError)
```

Description

Constructs a **Session** object; see **"Session Interface" on page 74**.

Parameters

Name	Type	Description
pXAQc	SBYN_Connection*	Pointer to the Connection object.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

SBYN_Session* pointer.

DeleteXAQueueConnectionFactory

Syntax

```
DeleteXAQueueConnectionFactory(pXAQcf, iError, pczError)
```

Description

Deletes the **ConnectionFactory** object.

Parameters

Name	Type	Description
pXAQcf	SBYN_XAQueueConnectionFactory*	Pointer to the ConnectionFactory.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

None.

4.25 XA Topic Session Methods

In XA pub/sub messaging, a client uses an XA topic session to create subscriber and publisher objects. The XA Topic Session includes the following methods:

XASessionGetTopicSession

Syntax

```
XASessionGetTopicSession(pXATs, iError, pczError)
```

Description

Retrieves the topic session associated with this XATopicSession.

Parameters

Name	Type	Description
pXATs	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_Session* pointer.

XASessionGetXAResource

Syntax

```
XASessionGetXAResource(p, iError, pczError)
```

Description

Retrieves the XA resource associated with this Session.

Parameters

Name	Type	Description
p	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_XAResource* pointer.

4.26 XA Queue Session Methods

In XA point-to-point messaging, a client uses an XA queue session to create sending and receiving objects. The XA Queue Session includes the following methods:

XASessionGetQueueSession

Syntax

```
XASessionGetQueueSession(pXAQs, iError, pczError)
```

Description

Retrieves the queue session associated with this Session.

Parameters

Name	Type	Description
pXAQs	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_Session* pointer.

4.27 XA Resource Methods

The XA Resource methods provide messaging instructions to the XA resource manager. They include the following:

DeleteXAResource

Syntax

```
DeleteXAResource(pXARc, iError, pczError)
```

Description

Deletes the XA resource.

Parameters

Name	Type	Description
pXARc	SBYN_XAResource*	Pointer to the XA resource.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

None.

XAResourceCommit

Syntax

```
XAResourceCommit(pXARc, pXid, pOnePhase, iError, pczError)
```

Description

Commits the global transaction specified by the global transaction identifier.

Parameters

Name	Type	Description
pXARc	SBYN_XAResource*	Pointer to the XA resource.
pXid	SBYN_Xid*	Pointer to the global transaction identifier.
pOnePhase	SBYN_BOOL	Flag; if true, a one-phase commit protocol is used to commit the work done on behalf of xid.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

None.

XAResourceRecover

Syntax

```
XAResourceRecover(pXARc, iFlag, iError, pczError)
```

Description

Used during recovery to obtain the list of transaction branches that are currently in prepared or heuristically completed states.

Parameters

Name	Type	Description
pXARc	SBYN_XAResource*	Pointer to the XA resource.
iFlag	int	Flag that specifies the recovery mode.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

SBYN_Xid*.

XAResourceRollback

Syntax

```
XAResourceRollback(pXARc, pXid, iError, pczError)
```

Description

Rolls back work done on behalf of a transaction branch.

Parameters

Name	Type	Description
pXARc	SBYN_XAResource*	Pointer to the XA resource.
pXid	SBYN_Xid*	Pointer to the global transaction identifier.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

None.

XAResourceGetTransactionTimeout

Syntax

```
XAResourceGetTransactionTimeout(pXARc, iError, pczError)
```

Description

Retrieves the current transaction timeout value set for this XA resource.

Parameters

Name	Type	Description
pXARc	SBYN_XAResource*	Pointer to the XA resource.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

int.

XAResourceSetTransactionTimeout

Syntax

```
XAResourceSetTransactionTimeout(pXARc, pSeconds, iError, pczError)
```

Description

Sets the transaction timeout value for this XA resource.

Parameters

Name	Type	Description
pXARc	SBYN_XAResource*	Pointer to the XA resource.
pSeconds	int	Number of seconds to time out.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

SBYN_BOOL.

XAResourceIsSameRM

Syntax

```
XAResourceIsSameRM(pXARc, pXARes, iError, pczError)
```

Description

Determines whether the specified resource managers are the same.

Parameters

Name	Type	Description
pXARc	SBYN_XAResource*	Pointer to the first XA resource.
pXARes	SBYN_XAResource*	Pointer to the second XA resource.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

SBYN_BOOL.

XAResourcePrepare

Syntax

```
XAResourcePrepare(pXARc, pXid, iError, pczError)
```

Description

Requests that the resource manager prepare for a transaction commit of the transaction specified by the global transaction identifier.

Parameters

Name	Type	Description
pXARc	SBYN_XAResource*	Pointer to the XA resource.
pXid	SBYN_Xid*	Pointer to the global transaction identifier.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

int.

XAResourceStart

Syntax

```
XAResourceStart(pXARc, pXid, flags, iError, pczError)
```

Description

Starts work on behalf of a transaction branch specified by the global transaction identifier.

Parameters

Name	Type	Description
pXARc	SBYN_XAResource*	Pointer to the XA resource.
pXid	SBYN_Xid*	Pointer to the global transaction identifier.
flags	int	Flag that indicates the start mode.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

None.

XAResourceEnd

Syntax

```
XAResourceEnd(pXARc, pXid, iFlags, iError, pczError)
```

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 to be completed.

Parameters

Name	Type	Description
pXARc	SBYN_XAResource*	Pointer to the first XA resource.
pXid	SBYN_Xid*	Pointer to the global transaction identifier.
iFlags	int	Flag that indicates the end mode.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

None.

4.28 XA Xid Methods

The XA Xid methods allow you to work with the global transaction identifier used for XA transactions. The methods include the following:

- **XACreateXid** on page 143
- **DeleteXid** on page 144
- **XIDGetBranchQualifier** on page 144
- **XIDGetFormatId** on page 145
- **XIDGetGlobalTransactionId** on page 145

XACreateXid

Syntax

```
XACreateXid(dll, iError, pczError)
```

Description

Creates an XID.

Parameters

Name	Type	Description
dll	char*	Pointer to a client library file. This should always be set to MSCLIENT_DLL_NAME unless you rename the default libraries shipped with the API Kit.
iError	int*	Pointer to the error number.

Name	Type	Description
pczError	char*	Pointer to the error message text.

Return Value

SBYN_Xid*.

DeleteXid

Syntax

```
DeleteXid(pXid, iError, pczError)
```

Description

Deletes an XID.

Parameters

Name	Type	Description
pXid	SBYN_Xid*	Pointer to the global transaction identifier.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

None.

XIDGetBranchQualifier

Syntax

```
XIDGetBranchQualifier(pXid, pl, iError, pczError)
```

Description

Retrieves the transaction branch identifier portion of the XID.

Parameters

Name	Type	Description
pXid	SBYN_Xid*	Pointer to the global transaction identifier.
pl	long	A long address.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

unsigned char*.

XIDGetFormatId

Syntax

```
XIDGetFormatId(pXid, iError, pczError)
```

Description

Retrieves the format identifier portion of the XID.

Parameters

Name	Type	Description
pXid	SBYN_Xid*	Pointer to the global transaction identifier.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

uint64_t.

XIDGetGlobalTransactionId

Syntax

```
XIDGetGlobalTransactionId(pXid, pl, iError, pczError)
```

Description

Retrieves the global transaction identifier portion of the XID.

Parameters

Name	Type	Description
pXid	SBYN_Xid*	Pointer to the global transaction identifier.
pl	long	A long address.
iError	int*	Pointer to the error number.
pczError	char*	Pointer to the error message text.

Return Value

unsigned char*.

4.29 Error Codes and Messages

The **C** API defines error codes as shown in Table 11.

Table 11 Error Codes Defined in the C API

Error Code	Explanation
0x80040300 = 2147746560 decimal JE_CODE_E_GENERAL	JMS exception, unspecified.
0x80040301 = 2147746561 decimal JE_CODE_E_REALLOC	A JMS exception occurred as a result of memory reallocation.
0x80040302 = 2147746562 decimal JE_CODE_E_MALLOC	A JMS exception occurred as a result of memory allocation.
0x80040303 = 2147746563 decimal JE_CODE_E_CONNECTION	A JMS exception occurred in setting up a connection.
0x80040304 = 2147746564 decimal JE_CODE_E_CREATION	A JMS exception occurred while creating a JMS object.
0x80040305 = 2147746565 decimal JE_CODE_E_CLOSED_SOCKET	A JMS exception occurred because of a closed socket.
0x80040306 = 2147746566 decimal JE_CODE_E_EOF	Processing ended because the BytesMessage or StreamMessage ended unexpectedly.
0x80040307 = 2147746567 decimal JE_CODE_E_NOTREADABLE	Processing ended because the message could not be read.
0x80040308 = 2147746568 decimal JE_CODE_E_NOTWRITEABLE	Processing ended because the message could not be written.
0x80040309 = 2147746569 decimal JE_CODE_E_FORMAT	Processing ended because the JMS client attempted to use a data type not supported by the message (or a message property), or attempted to read message data (or a message property) as the wrong type.
0x8004030A = 2147746570 decimal JE_CODE_E_ROLLBACK	The attempt to commit the session was unsuccessful of a transaction being rolled back.
0x8004030B = 2147746571 decimal JE_CODE_E_STATE	Processing ended because a method was invoked at an illegal or inappropriate time or because the provider was not in an appropriate state for the requested operation.
0x8004030C = 2147746572 decimal JE_CODE_E_DESTINATION	Processing ended because the destination could not be understood or was found to be invalid.
0x8004030D = 2147746573 decimal JE_CODE_E_NOTIMPL	Processing ended because a feature or interface was not implemented.
0x8004030E = 2147746574 decimal JE_CODE_E_INDEX_OUT_OF_BOUNDS	Processing ended because an index of some sort (such as to an array, to a string, or to a vector) was found to be outside the valid range.
0x8004030F = 2147746575 decimal JE_CODE_E_NULL_POINTER	Processing ended because the pointer in a case where an object was required.
0x80040310 = 2147746576 decimal JE_CODE_E_INVLD_CLIENT_ID	Processing ended because the connection's client ID was rejected by the provider.

Table 11 Error Codes Defined in the C API

Error Code	Explanation
0x80040311 = 2147746577 decimal JE_CODE_E_INVLD_SELECTOR	Processing ended because the message selector was found to be syntactically invalid.
0x80040312 = 2147746578 decimal JE_CODE_E_JMS_SECURITY	Processing was ended by JMS Security — for example, the provider rejected a name/password combination submitted by a client.
0x80040313 = 2147746579 decimal JE_CODE_E_RESOURCE_ALLOC	Processing ended because of the provider was unable to allocate resources required for the method/function.
0x80040314 = 2147746580 decimal JE_CODE_E_XA_IN_PROGRESS	Processing ended because a transaction was in progress.

Chapter 5

Developing Java CAPS JMS Applications in C++

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

What's in This Chapter

- **Error Codes and Messages** on page 243

5.1 About the Java CAPS JMS Interface for Applications in C++

This section provides information about the C++ API for the JMS interface, including wrapper functions and mapping to the JMS API. The full C++ API reference begins with **"Message Interface" on page 152**.

5.1.1 Wrapper Functions for C++ Applications

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

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

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

- **Session**—Characterized by a hostname, port number, session type (either a pub/sub "topic session" or a point-to-point "queue session"), and connection parameters such as acknowledgment mode, transaction mode, delivery mode, and client ID.
- **Destination**—Either a topic (for pub/sub messaging) or else a queue (for point-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.

5.1.2 C++ API Mapping to the JMS API

Wherever possible, the C++ API was designed to correspond with standard JMS Java classes. For example, the Message and BytesMessage interfaces in the C++ API provide similar functionality to the JMS Message and BytesMessage interfaces. However, there are some differences between the C++ API and the JMS Java API. Table 12 lists the differences between C++ types and Java types.

Table 12 C++ Type Mapping to Java Types

C++ Type	Java Type
int64_t	long
int	int
unsigned char	byte
unsigned char *	bytes[]
float	float
double	double
WString	String
short	short
ucs2_t	char
SerialObject	Serializable

Other differences between the two APIs include the following.

- The **setText** method in the TextMessage Interface assumes a 0-terminated wide C string and has been overloaded with WString and char* (char* values are used internally and are converted to WString).
- In the Session interface, **ObjectMessage** is not implemented.
- In the TopicConnection interface, the **setClientId** method is overloaded to accept char* instead of String.
- The MessageProducer interface includes the methods **setJMS_ProducerID** and **getJMS_ProducerID**.
- In the TopicPublisher interface, the **publish** methods are overloaded for different message types.

5.2 Creating Destinations

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

5.3 C++ Constants

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

DeliveryMode Constants

Table 13 Values for DeliveryMode Constants

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

Session Constants

Table 14 Session Constants

Name	Value	Description
AUTO_ACKNOWLEDGE	1	**1** indicates auto-acknowledgment. The session automatically acknowledges a client's receipt of a message either upon its successful return from a call to **receive** or upon successful return of the MessageListener it has called to process the message.
CLIENT_ACKNOWLEDGE	2	**2** indicates acknowledgment by client: A client acknowledges a message by calling the message's **acknowledge** method. Acknowledging a consumed message automatically acknowledges the receipt of all messages that have been delivered by its session.

Table 14 Session Constants

Name	Value	Description
DUPS_OK_ACKNOWLEDGE	3	**3** indicates that duplicates are acceptable, and instructs the session to lazily acknowledge message delivery This setting is likely to cause delivery of some duplicate messages if JMS fails, and should only be used by consumers that are tolerant of duplicate messages. The benefit is the reduction of session overhead, achieved by minimizing the work done to prevent duplicates.

Message Default Constants

Table 15 Message Default Constants

Name	Value	
DEFAULT_DELIVERY_MODE	1	See **"DeliveryMode Constants" on page 151**.
DEFAULT_PRIORITY	4	JMS defines a ten-level priority value: **0** is lowest priority (least expedited) and **9** is highest. Clients should consider priorities **0** through **4** as gradations of normal priority and priorities **5** through **9** as gradations of expedited priority.
DEFAULT_TIME_TO_LIVE	0	Length of time that a produced message should be retained by the message system. Measured in milliseconds elapsed since its dispatch time. The default, **0**, has the special meaning of "retain forever" — that is, the message never expires on its own.

5.4 Message Interface

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

Most message-oriented middle ware (MOM) products treat messages as lightweight entities that consist of a header and a payload. The header contains fields used for message routing and identification; the payload contains the application data being sent.

Within this general form, the definition of a message varies significantly across products. It would be quite difficult for the JMS API to support all of these message models.

With this in mind, the JMS message model has the following goals:

- Provide a single, unified message API
- Provide an API suitable for creating messages that match the format used by provider-native messaging applications
- Support the development of heterogeneous applications that span operating systems, machine architectures, and computer languages
- Support messages containing objects in the Java programming language ("Java objects")
- Support messages containing Extensible Markup Language (XML) pages

The Message methods are:

acknowledge

Syntax

```
void acknowledge()
```

Description

Acknowledges the receipt of current and previous messages.

Return Value

None.

clearBody

Syntax

```
void clearBody()
```

Description

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

Return Value

None.

clearProperties

Syntax

```
void clearProperties()
```

Description

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

Return Value

None.

propertyExists

Syntax

```
STCBOOL propertyExists(name)
```

Description

Checks whether a value for a specific property exists.

Parameters

Name	Type	Description
name	WString&	The name of the property to check.

Return Value

STCBOOL

Returns **true** if the property value is defined; otherwise, returns **false**.

getBooleanProperty

Syntax

```
STCBOOL getBooleanProperty(name)
```

Description

Checks whether a value for a specific property exists.

Parameters

Name	Type	Description
name	WString&	The name of the property to check.

Return Value

STCBOOL

Returns **true** if the property value is defined; otherwise, returns **false**.

getByteProperty

Syntax

```
unsigned getByteProperty(name)
```

Description

Reads the value of the specified byte property in the specified message.

Parameters

Name	Type	Description
named	WString&	The name of the property to check.

Return Value

unsigned char

The value of the property.

getDoubleProperty

Syntax

```
double getDoubleProperty(name)
```

Description

Reads the value of the specified double (numeric) property in the specified message.

Parameters

Name	Type	Description
name	WString&	The name of the property to check.

Return Value

Double

The value of the property.

getFloatProperty

Syntax

```
float getFloatProperty(name)
```

Description

Reads the value of the specified floating-point (numeric) property in the specified message.

Parameters

Name	Type	Description
name	WString&	The name of the property to check.

Return Value

float

The value of the property.

getIntProperty

Syntax

```
int getIntProperty(name)
```

Description

Reads the value of the specified integer (numeric) property in the specified message.

Parameters

Name	Type	Description
name	WString&	The name of the property to check.

Return Value

int

The value of the property.

getLongProperty

Syntax

```
int64_t getLongProperty(name)
```

Description

Reads the value of the specified long (numeric) property in the specified message.

Parameters

Name	Type	Description
name	WString&	The name of the property to check.

Return Value

int64_t

The long value of the property.

getPropertyName

Syntax

```
wNamesList getPropertyName(name)
```

Description

Returns a list of WStrings for the specified message.

Parameters

Name	Type	Description
name	wNamesList	Returns a list of the property names.

Return Value

wNamesList

Returns a list.

getShortProperty

Syntax

```
short getShortProperty(name)
```

Description

Reads the value of the specified short (numeric) property in the specified message.

Parameters

Name	Type	Description
name	WString&	The name of the property to check.

Return Value

short

The value of the property.

getStringProperty

Syntax

```
WString getStringProperty(name)
```

Description

Reads the value of the specified text property in the specified message.

Parameters

Name	Type	Description
name	WString	The name of the property to check.

Return Value

WString

A WString (wide string) object containing the value of the property.

setBooleanProperty

Syntax

```
void setBooleanProperty(name, value)
```

Description

Writes a value for the specified Boolean property in the specified message.

Parameters

Name	Type	Description
name	WString&	The name of the property to check.
value	STCBool	The value to be written.

Return Value

None

setByteProperty

Syntax

```
void setByteProperty(name, value)
```

Description

Writes a value for the specified byte property in the specified message.

Parameters

Name	Type	Description
name	WString&	The name of the property to check.
value	unsigned char	The value to be written.

Return Value

None

setDoubleProperty

Syntax

```
void setDoubleProperty(name, value)
```

Description

Writes a value for the specified double (numeric) property in the specified message.

Parameters

Name	Type	Description
name	WString&	The name of the property to check.
value	double	The value to be written.

Return Value

None

setFloatProperty

Syntax

```
void setFloatProperty(name, value)
```

Description

Writes a value for the specified byte property in the specified message.

Parameters

Name	Type	Description
name	WString&	The name of the property to check.
value	float	The value to be written.

Return Value

None

setIntProperty

Syntax

```
void setIntProperty(name, value)
```

Description

Writes a value for the specified byte property in the specified message.

Parameters

Name	Type	Description
name	WString&	The name of the property to check.
value	int	The value to be written.

Return Value

None

setLongProperty

Syntax

```
void setLongProperty(name, value)
```

Description

Writes a value for the specified long (numeric) property in the specified message.

Parameters

Name	Type	Description
name	WString&	The name of the property to check.
User defined	long	The value to be written.

Return Value

None

setObjectProperty

Syntax

```
void setObjectProperty(name, value)
```

Description

Writes a value for the specified object property in the specified message.

Parameters

Name	Type	Description
name	WString&	The name of the property to check.
User defined	SerialObject	The value to be written.

Return Value

None

setShortProperty

Syntax

```
void setShortProperty(name, value)
```

Description

Writes a value for the specified short (numeric) property in the specified message.

Parameters

Name	Type	Description
name	WString&	The name of the property to check.
value	short	The value to be written.

Return Value

None

setStringProperty

Syntax

```
void setStringProperty(name, value)
```

Description

Writes a value for the specified text property in the specified message.

Parameters

Name	Type	Description
name	WString&	The name of the property to check.
value	char *	The value to be written.

Return Value

None

propertyExists

Syntax

```
STCBOOL propertyExists(name)
```

Description

Checks whether a value for a specific property exists.

Parameters

Name	Type	Description
name	char*	Pointer to the name of the property to check.

Return Value

STCBOOL

Returns **true** if the property value is defined; otherwise, returns **false**.

getBooleanProperty

Syntax

```
STCBOOL getBooleanProperty(name)
```

Description

Reads the value of the specified Boolean property in the specified message.

Parameters

Name	Type	Description
name	char*	Pointer to the name of the property to check.

Return Value

STCBOOL
The value of the property.

getByteProperty

Syntax

```
unsigned char getByteProperty(name)
```

Description

Reads the value of the specified byte property in the specified message.

Parameters

Name	Type	Description
name	char*	Pointer to the name of the property to check.

Return Value

unsigned char
The value of the property.

getDoubleProperty

Syntax

```
double getDoubleProperty(name)
```

Description

Reads the value of the specified double (numeric) property in the specified message.

Parameters

Name	Type	Description
name	char*	Pointer to the name of the property to check.

Return Value

double
The value of the property.

getFloatProperty

Syntax

```
float getFloatProperty(name)
```

Description

Reads the value of the specified floating-point (numeric) property in the specified message.

Parameters

Name	Type	Description
name	char*	Pointer to the name of the property to check.

Return Value

float
The value of the property.

getIntProperty

Syntax

```
int getIntProperty(name)
```

Description

Reads the value of the specified integer (numeric) property in the specified message.

Parameters

Name	Type	Description
name	char*	Pointer to the name of the property to check.

Return Value

int
The value of the property.

getLongProperty

Syntax

```
int64_t getLongProperty(name)
```

Description

Reads the value of the specified long (numeric) property in the specified message.

Parameters

Name	Type	Description
name	char*	Pointer to the name of the property to check.

Return Value

int64_t
The long value of the property.

getObjectProperty

Syntax

```
SerialObject getObjectProperty(name)
```

Description

Reads the value of the specified object property in the specified message.

Parameters

Name	Type	Description
name	char*	Pointer to the name of the object property to check.

Return Value

SerialObject
The serialized value of the property.

getShortProperty

Syntax

```
short getShortProperty(name)
```

Description

Reads the value of the specified short (numeric) property in the specified message.

Parameters

Name	Type	Description
name	char*	Pointer to the name of the property to check.

Return Value

short
The value of the property.

getStringProperty

Syntax

```
WString getStringProperty(name)
```

Description

Reads the value of the specified text property in the specified message.

Parameters

Name	Type	Description
name	char*	Pointer to the name of the property to check.

Return Value

WString*
Pointer to a **WString** (wide string) object containing the value of the property.

setBooleanProperty

Syntax

```
void setBooleanProperty(name, value)
```

Description

Writes a value for the specified Boolean property in the specified message.

Parameters

Name	Type	Description
name	char*	Pointer to the property name.
value	STCBOOL	The value to be written.

Return Value

None.

setByteProperty

Syntax

```
void setByteProperty(name, value)
```

Description

Writes a value for the specified byte property in the specified message.

Parameters

Name	Type	Description
name	char*	Pointer to the property name.
value	unsigned char	The value to be written.

Return Value

None.

setDoubleProperty

Syntax

```
void setDoubleProperty(name, value)
```

Description

Writes a value for the specified double (numeric) property in the specified message.

Parameters

Name	Type	Description
name	char*	Pointer to the property name.
value	double	The value to be written.

Return Value

None.

setFloatProperty

Syntax

```
void setFloatProperty(name, value)
```

Description

Writes a value for the specified floating-point (numeric) property in the specified message.

Parameters

Name	Type	Description
name	char*	Pointer to the property name.
Value	float	The value to be written.

Return Value

None.

setIntProperty

Syntax

```
void setIntProperty(name, value)
```

Description

Writes a value for the specified integer (numeric) property in the specified message.

Parameters

Name	Type	Description
name	char*	Pointer to the property name.
value	int	The value to be written.

Return Value

None.

setLongProperty

Syntax

```
void setLongProperty(name, value)
```

Description

Writes a value for the specified long (numeric) property in the specified message.

Parameters

Name	Type	Description
name	char*	Pointer to the property name.
value	int64_t	The value to be written.

Return Value

None.

setObjectProperty

Syntax

```
void setObjectProperty(name, value)
```

Description

Writes a value for the specified long (numeric) property in the specified message.

Parameters

Name	Type	Description
name	char*	Pointer to the property name.
value	SerialObject	The value to be written.

Return Value

None.

setShortProperty

Syntax

```
void setShortProperty(name, value)
```

Description

Writes a value for the specified short (numeric) property in the specified message.

Parameters

Name	Type	Description
name	char*	Pointer to the property name.
value	short	The value to be written.

Return Value

None.

setStringProperty

Syntax

```
void setStringProperty(name, value)
```

Description

Writes a value for the specified text property in the specified message.

Parameters

Name	Type	Description
name	char*	Pointer to the property name.
value	char*	The value to be written.

Return Value

None.

getJMSCorrelationID

Syntax

```
WString getJMSCorrelationID()
```

Description

Retrieves the correlation ID for the specified message.

Return Value

WString*
Pointer to a **WString** (wide string) object containing the text.

getJMSCorrelationIDAsBytes

Syntax

```
char* getJMSCorrelationIDAsBytes()
```

Description

Retrieves the correlation ID for the specified message as an array of characters.

Return Value

None

getJMSDeliveryMode

Syntax

```
int getJMSDeliveryMode()
```

Description

Retrieves the value of the **DeliveryMode** property for the specified message.

Return Value

None

getJMSExpiration

Syntax

```
int64_t getJMSExpiration()
```

Description

Retrieves the value of the timestamp set for the expiration of the specified message.

Return Value

int64_t
Long timestamp.

getJMSMessageID

Syntax

```
WString getJMSMessageID()
```

Description

Retrieves the message ID of the specified message.

Return Value

WString*
Pointer to a **WString** (wide string) object containing the text.

getJMSPriority

Syntax

```
int getJMSPriority()
```

Description

Retrieves the priority level for the specified message.

Return Value

int
The priority level.

getJMSRedelivered

Syntax

```
STCBOOL getJMSRedelivered()
```

Description

Retrieves an indication of whether the specified message is being redelivered.

Return Value

STCBOOL
Returns **true** if the message is being redelivered; otherwise, returns **false**.

getJMSReplyTo

Syntax

```
Destination* getJMSReplyTo()
```

Description

Retrieves the **Destination** object where a reply to the specified message should be sent (for request/reply messaging).

Return Value

SBYN_Destination*
Pointer to the **Destination** object.

getJMSTimestamp

Syntax

```
int64_t getJMSTimestamp()
```

Description

Retrieves the timestamp of the specified message.

Return Value

int64_t
Long timestamp.

getJMSType

Syntax

```
WString getJMSType()
```

Description

Gets the message type identifier supplied by the client when the message was sent.

Return Value

WString*
Pointer to a **WString** (wide string) object containing the text.

setJMSCorrelationID

Syntax

```
void setJMSCorrelationID(correlationID)
```

Description

Sets the correlation ID for the specified message.

Parameters

Name	Type	Description
correlationID	WString	The text string containing the correlation ID.

Return Value

None.

setJMSCorrelationIDAsBytes

Syntax

```
void setJMSCorrelationIDAsBytes(deliveryMode)
```

Description

Sets the correlation ID for the specified message.

Parameters

Name	Type	Description
deliveryMode	char *	Pointer to the character array containing the bytes for the correlation ID.

Return Value

None.

setJMSDeliveryMode

Syntax

```
void setJMSDeliveryMode(destination)
```

Description

Sets the delivery mode for the specified message.

Parameters

Name	Type	Description
destination	Destination*	Pointer to the value corresponding to the delivery mode.

Return Value

None.

setJMSExpiration

Syntax

```
void setJMSExpiration(expiration)
```

Description

Sets the timestamp at which the specified message is due to expire.

Parameters

Name	Type	Description
expiration	int64_t	Timestamp of the expiration.

Return Value

None.

setJMSMessageID

Syntax

```
void setJMSMessageID(id)
```

Description

Sets the message ID of the specified message.

Parameters

Name	Type	Description
id	WString	The text string containing the message ID.

Return Value

None.

setJMSPriority

Syntax

```
void setJMSPriority(priority)
```

Description

Sets the priority level for the specified message.

Parameters

Name	Type	Description
priority	int	The priority level.

Return Value

None.

setJMSRedelivered

Syntax

```
void setJMSRedelivered(redelivered)
```

Description

Determines whether to flag the specified message as being redelivered. Used, for example, to specify re-delivery for a message that has been sent but not acknowledged.

Parameters

Name	Type	Description
redelivered	STCBOOL	Flag: If **true**, the message is being redelivered.

Return Value

None.

setJMSReplyTo

Syntax

```
void setJMSReplyTo(replyTo)
```

Description

Sets the **Destination** object where a reply to the specified message should be sent.

Parameters

Name	Type	Description
replyto	Destination*	Pointer to the **Destination** object.

Return Value

None.

setJMSTimestamp

Syntax

```
void setJMSTimestamp(timestamp)
```

Description

Sets the timestamp (JMSTimestamp header field) that the specified message was handed off to a provider to be sent. Note that this is not necessarily the time the

message is actually transmitted; the actual **send** can occur later because of transactions or other client-side queueing of messages.

Parameters

Name	Type	Description
timestamp	int64_t	Timestamp to be set.

Return Value

None.

setJMSType

Syntax

```
void setJMSType(type)
```

Description

Sets the JMSType header field, which is often used to identify the message structure and the payload type.

Parameters

Name	Type	Description
type	WString	The text string containing the data.

Return Value

None.

setJMSCorrelationIDAsBytes

Syntax

```
void setJMSCorrelationIDAsBytes(correlationID)
```

Description

Sets the correlation ID for the specified message.

Parameters

Name	Type	Description
correlationID	char*	Pointer to the character array containing the bytes for the correlation ID.

Return Value

None.

setJMSMessageID

Syntax

```
void setJMSMessageID(id)
```

Description

Sets the message ID of the specified message.

Parameters

Name	Type	Description
id	char*	Pointer to the text string containing the message ID.

Return Value

None.

setJMSType

Syntax

```
void setJMSType(type)
```

Description

Sets the JMSType header field, which is often used to identify the message structure and the payload type.

Parameters

Name	Type	Description
pczJMSType	char*	Pointer to the text string containing the data.

Return Value

None.

5.5 BytesMessage Interface

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

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.

writeBytesEx

Syntax

```
WriteBytesEx(value, offset, length)
```

Description

Writes a portion of a byte array (unsigned char values) to the **BytesMessage** stream. For example, to extract “nag” from “manager”, set iOffset=2 and iLength=3.

Parameters

Name	Type	Description
pczValue	unsigned char*	Pointer to the value to be written.
offset	int	The initial offset within the byte array.
length	int	The number of bytes to use.

Return Value

None.

writeChar

Syntax

```
void writeChar(value)
```

Description

Writes an unsigned short integer to the **BytesMessage** stream.

Parameters

Name	Type	Description
value	unsigned short	The value to be written.

Return Value

None.

writeDouble

Syntax

```
void writeDouble(value)
```

Description

Writes a double numeric value to the **BytesMessage** stream.

Parameters

Name	Type	Description
value	double	The value to be written.

Return Value

None.

writeFloat

Syntax

```
writeFloat(value)
```

Description

Writes a floating-point numeric value to the **BytesMessage** stream.

Parameters

Name	Type	Description
value	float	The value to be written.

Return Value

None.

writeInt

Syntax

```
void writeInt(value)
```

Description

Writes an integer numeric value to the **BytesMessage** stream.

Parameters

Name	Type	Description
value	int	The value to be written.

Return Value

None.

writeLong

Syntax

```
void writeLong(value)
```

Description

Writes a long numeric value to the **BytesMessage** stream.

Parameters

Name	Type	Description
value	int	The value to be written.

Return Value

None.

writeShort

Syntax

```
void writeShort(value)
```

Description

Writes a short numeric value to the **BytesMessage** stream.

Parameters

Name	Type	Description
value	short	The value to be written.

Return Value

None.

5.6 The MapMessage Class

A MapMessage is used to send a set of name-value pairs, where names are Strings. The entries are accessed sequentially or randomly by name. The order of the entries is undefined. It inherits from Message, and adds a map message body.

The methods of the MapMessage Class are:

getBoolean

Syntax

```
STCBOOL getBoolean(name)
```

Description

The getBoolean method returns the boolean value with the given name

Parameters

Name	Type	Description
name	WString	The name of the boolean property.

Return Value

STCBOOL

getByte

Syntax

```
unsigned char getByte(name)
```

Description

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

Parameters

Name	Type	Description
name	WString	The name of the byte.

Return Value

unsigned char

getBytes

Syntax

```
unsigned char* getBytes(name)
```

Description

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

Parameters

Name	Type	Description
name	WString	The name of the byte array value.

Return Value

unsigned char*
Pointer to the byte array value.

getBytes

Syntax

```
unsigned int getBytes(name, value, length)
```

Description

The getBytes method returns the int value with the given name as a variable.

Parameters

Name	Type	Description
name	WString	The name of the byte array value.
value	int	Value to be written.
length	int	The number of bytes to use.

Return Value

unsigned int
The value read.

getChar

Syntax

```
wchar_t getChar(name)
```

Description

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

Parameters

Name	Type	Description
name	WString	The name of the Unicode character.

Return Value

wchar_t

getDouble

Syntax

```
double getDouble(name)
```

Description

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

Parameters

Name	Type	Description
name	WString	The name of the double property.

Return Value

double

getFloat

Syntax

```
float getFloat(name)
```

Description

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

Parameters

Name	Type	Description
name	WString	The name of the float property.

Return Value

float

getInt

Syntax

```
int getInt(name)
```

Description

The getInt method returns the int value of the given name

Parameters

Name	Type	Description
name	WString	The value of the name.

Return Value

int

getLong

Syntax

```
int64_t getLong(name)
```

Description

The getLong method returns the long value of the given name.

Parameters

Name	Type	Description
name	WString	The long value of the name.

Return Value

long

getObject

Syntax

```
getObject(name)
```

Description

The getObject method returns a copy of the Java object value with the given name in objectified format. Note that byte values are returned as byte[], not Byte[].

Parameters

Name	Type	Description
name	WString	Name of the object.

Return Value

WString, or null if there is no item by the given name.

getShort

Syntax

```
short getShort (name)
```

Description

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

Parameters

Name	Type	Description
name	WString	The name of the short property.

Return Value

short

getString

Syntax

```
WString getString(name)
```

Description

The getString method returns the String value with the given name

Parameters

Name	Type	Description
name	WString	The name of the String property.

Return Value

Wstring

itemExists

Syntax

```
STCBOOL itemExists(name)
```

Description

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

Parameters

Name	Type	Description
name	WString	The name of the item to check.

Return Value

An **STCBOOL** indicator of whether the item exists.

setBoolean

Syntax

Description

The setBoolean method sets a boolean property value with the given name into the MapMessage.

Parameters

```
void setBoolean (name, value)
```

Name	Type	Description
name	WString	The name of the property value.

Name	Type	Description
value	STCBOOL	The value to set in the message.

Return Value

None.

setByte

Syntax

```
void setByte(name, value)
```

Description

The setByte method sets a byte value with the given name into the MapMessage.

Parameters

Name	Type	Description
name	WString	The name of the byte property.
value	unsigned char	The byte property value to set in the message.

Return Value

None.

setBytes

Syntax

```
void setBytes(name, value, [offset], [length])
```

Description

The setBytes method sets a byte array or a portion of value with the given name into the MapMessage.

Parameters

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

Return Value

None.

setChar

Syntax

```
void setChar(name, value)
```

Description

The setChar method sets a Unicode character value with the given name into the MapMessage.

Parameters

Name	Type	Description
name	WString	The name of the Unicode character.
value	ucs2_t	The Unicode character value to set in the Map.

Return Value

None.

setDouble

Syntax

```
void setDouble(name, value)
```

Description

The setDouble method sets a double value with the given name into the Map Message.

Parameters

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

Return Value

None.

setFloat

Syntax

```
void setFloat(name, value)
```

Description

The setFloat method sets a float value with the given name into the MapMessage.

Parameters

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

Return Value

None.

setInt

Syntax

```
void setInt(name, value)
```

Description

The setInt method sets a long value with the given name into the Map Message.

Parameters

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

Return Value

None.

setLong

Syntax

```
void setLong(name, value)
```

Description

The setLong method sets a currency value with the given name into the MapMessage.

Parameters

Name	Type	Description
name	WString	The name of the currency property.
value	int64_t	The currency property value to set in the message.

Return Value

None.

setObject

Syntax

```
void setObject(name, value)
```

Description

The setObject method sets an object value with the given name into the MapMessage. This method only works for the objectified primitive object types, Strings and byte arrays.

Parameters

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

Return Value

None.

setShort

Syntax

```
setShort(name, value)
```

Description

The setShort method sets a short value with the given name into the MapMessage.

Parameters

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

Return Value

None.

setString

Syntax

```
void setString(name, value)
```

Description

The setString method sets a String value with the given name into the MapMessage.

Parameters

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

Return Value

None.

5.7 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 195
- **setText** on page 195
- **setText** on page 196

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.

5.8 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:

- **close** on page 196
- **getClientID** on page 197
- **setClientID** on page 197
- **start** on page 197
- **stop** on page 198

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.

5.9 QueueConnection Interface

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

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.

5.10 Session Interface

The Session Interface methods:

close

Syntax

```
void close()
```

Description

Closes the specified session.

Note: *Sessions should be closed when they are no longer needed.*

Return Value

None.

commit

Syntax

```
void commit()
```

Description

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

Return Value

None.

getTransacted

Syntax

```
STCBOOL getTransacted()
```

Description

Queries whether the specified session is or is not transacted.

STCBOOL

Returns **true** if the session is transacted; otherwise, returns **false**.

recover

Syntax

```
void recover()
```

Description

Stops message delivery in the specified session, causes all messages that might have been delivered but not acknowledged to be marked as **redelivered**, and restarts message delivery with the oldest unacknowledged message. Note that redelivered messages need not be delivered in the exact order they were originally delivered.

Return Value

None.

rollback

Syntax

```
rollback(pSessn, iError, pczError)
```

Description

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

Return Value

None.

bytesMessage

Syntax

```
BytesMessage* bytesMessage()
```

Description

Creates a **BytesMessage**— an object used to send a message containing a stream of uninterpreted bytes.

Return Value

BytesMessage*
Pointer to the created message object.

createTextMessage

Syntax

```
TextMessage createTextMessage()
```

Description

Creates an uninitialized **TextMessage**— an object used to send a message containing a string to be supplied.

Return Value

TextMessage
The created message object.

---

createTextMessage

Syntax

```
TextMessage createTextMessage(stringBuffer)
```

Description

Creates an initialized **TextMessage**— an object used to send a message containing the supplied string.

Parameters

Name	Type	Description
stringBuffer	WString	stringBuffer to the Text message.

Return Value

TextMessage

5.11 TopicConnection Interface

The TopicConnection Interface methods are:

---

createTopicSession

Description

```
ConnectionConsumer createTopicSession(transacted, acknowledgeMode)
```

Syntax

Create a TopicSession

Parameters

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

Return Value

ConnectionConsumer

close

Syntax

```
void close()
```

Description

Closes the specified session.

Note: *Sessions should be closed when they are no longer needed.*

Return Value

None.

getClientID

Syntax

```
WString getClientID()
```

Description

Retrieves the client ID associated with the specified **Connection** object.

Return Value

WString*
Pointer to a **WString** (wide string) object containing the text.

setClientID

Syntax

```
void setClientID(clientID)
```

Description

Sets the client ID to be associated with the specified **Connection** object. In other words, this method allows you to name the connection.

Parameters

Name	Type	Description
clientID	void	The text string for the client ID.

Return Value

None.

setClientID

Syntax

```
void setClientID(clientID)
```

Description

Sets the client ID to be associated with the specified **Connection** object. In other words, this method allows you to name the connection.

Parameters

Name	Type	Description
clientID	char*	Pointer to the text string for the client ID.

Return Value

char *
Pointer to the text string to the client ID.

getExceptionListener

Syntax

ExceptionListener getExceptionListener()

Description

Gets the **ExceptionListener** object for this connection.

Return Value

ExceptionListener

5.12 QueueConnectionFactory Interface

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 204
- **createQueueConnection** on page 204

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 **QueueConnection** object.

Parameters

Name	Type	Description
userName	char*	Pointer to the userName.
password	char*	Pointer to the password.

Return Value

QueueConnection*
Pointer to the **QueueConnection** object that was created.

5.13 TopicConnectionFactory Interface

Using pub/sub messaging, a client uses a **TopicConnectionFactory** object to create **TopicConnection** objects.

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

createTopicConnection

Syntax

```
TopicConnection* createTopicConnection()
```

Description

Constructs a **TopicConnection** object.

Return Value

TopicConnection*
Pointer to the **TopicConnection** object that was created.

createTopicConnection

Syntax

```
TopicConnection createTopicConnection(userName, password)
```

Description

Constructs a **TopicConnection** object.

Parameters

Name	Type	Description
userName	char*	Pointer to the userName.
password	char*	Pointer to the password.

Return Value

TopicConnection*
Pointer to the **TopicConnection** object that was created.

5.14 ExceptionListener Interface

If the JMS IQ manager detects a serious problem with a Connection object, it informs the Connection object's ExceptionListener, if one has been registered. It does this by calling the listener's onException method, passing it a JMSException argument describing the problem.

This allows a client to be asynchronously notified of a problem. Some Connections only consume messages so they would have no other way to learn their Connection has failed.

A JMS provider should attempt to resolve connection problems themselves prior to notifying the client of them.

The ExceptionListener Interface method is:

- **OnException** on page 206

OnException

Syntax

```
void OnException(exception)
```

Description

```
Notifies user of a JMS exception.
```

Parameters

Name	Description
exception	The JMS exception.

Return Value

None.

5.15 DeliveryMode Interface

The delivery modes supported by the JMS API are:

- **NON_PERSISTENT Field** on page 207
- **PERSISTENT Field** on page 207

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.

5.16 Queue Interface

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

Syntax

```
WString getQueueName()
```

Description

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

Return Value

WString
Wide string object.

toString

Syntax

```
WString toString()
```

Description

Return a pretty printed version of the queue name

Return Value

WString
Wide string object.

5.17 TemporaryQueue Interface

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

Syntax

```
void Delete()
```

Description

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

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

Return Value

None.

5.18 Topic Interface

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

Syntax

WString getTopicName()

Description

Gets the name of this topic.

Return Value

WString
Wide string object.

toString

Syntax

WString **toString**()

Description

Returns a string representation of this object.

Return Value

WString
Wide string object.

5.19 TemporaryTopic Interface

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 210

Delete

Syntax

void Delete()

Description

Deletes this temporary topic. If there are existing subscribers still using it, a `JMSException` will be thrown.

Return Value

None.

5.20 MessageProducer Interface

The **MessageProducer** interface includes the following methods:

- **close** on page 210
- **getDeliveryMode** on page 211
- **getDisableMessageID** on page 211
- **getDisableMessageTimestamp** on page 211
- **getJMS_ProducerID** on page 211
- **getPriority** on page 212
- **getTimeToLive** on page 212
- **setDeliveryMode** on page 212
- **setDisableMessageID** on page 213
- **setDisableMessageTimestamp** on page 211
- **setJMS_ProducerID** on page 213
- **setPriority** on page 214
- **setTimeToLive** on page 214

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.

5.21 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:

send

Syntax

```
void send(message)
```

Description

Sends the specified message to the queue.

Parameters

Name	Type	Description
message	Message*	Pointer to the message to send.

Return Value

None.

send

Syntax

```
void send(message)
```

Description

Sends the specified message to the queue.

Parameters

Name	Type	Description
message	BytesMessage*	Pointer to the message to send.

Return Value

None.

send

Syntax

```
void send(message)
```

Description

Sends the specified message to the queue.

Parameters

Name	Type	Description
message	TextMessage*	Pointer to the message to send.

Return Value

None.

send

Syntax

```
void send(message, DeliveryMode, priority, timeToLive)
```

Description

Sends the specified message to the queue, overriding one or more default values for properties of the specified queue sender.

Parameters

Name	Type	Description
message	message*	Pointer to the message to send.
deliveryMode	int	Value to be used for **DeliveryMode.**
priority	int	Value to be used for **Priority**.
timeToLive	int64_t	Value to be used for **TimeToLive**: Length of time (milliseconds elapsed since dispatch time) that a produced message should be retained by JMS.

Return Value

None.

send

Syntax

```
void send(message, DeliveryMode, priority, timeToLive)
```

Description

Sends the specified message to the queue, overriding one or more default values for properties of the specified queue sender.

Parameters

Name	Type	Description
message	BytesMessage*	Pointer to the message to send.
deliveryMode	int	Value to be used for **DeliveryMode.**
priority	int	Value to be used for **Priority**.
timeToLive	int64_t	Value to be used for **TimeToLive**: Length of time (milliseconds elapsed since dispatch time) that a produced message should be retained by JMS.

Return Value

None.

send

Syntax

```
void send(message, DeliveryMode, priority, timeToLive)
```

Description

Sends the specified message to the queue, overriding one or more default values for properties of the specified queue sender.

Parameters

Name	Type	Description
message	TextMessage*	Pointer to the message to send.
deliveryMode	int	Value to be used for **DeliveryMode.**
priority	int	Value to be used for **Priority**.
timeToLive	int64_t	Value to be used for **TimeToLive**: Length of time (milliseconds elapsed since dispatch time) that a produced message should be retained by JMS.

Return Value

None.

send

Syntax

```
void send(queue, message)
```

Description

Sends the specified message to the specified queue, using default values for properties (**DeliveryMode**, **Priority**, and **TimeToLive**) of the specified queue sender.

Typically, a message producer is assigned a queue at creation time; however, the JMS API also supports unidentified message producers, which require that the queue be supplied every time a message is sent.

Parameters

Name	Type	Description
queue	Queue*	Pointer to the **QueueSender** object.
message	Message*	Pointer to the message to send.

Return Value

None.

send

Syntax

```
send(queue, message, deliveryMode, priority, timeToLive)
```

Description

Sends the specified message to the specified queue, overriding one or more default values for properties of the specified queue sender.

Parameters

Name	Type	Description
queue	Queue*	Pointer to the **QueueSender** object.
message	Message*	Pointer to the message to send.
deliveryMode	int	Value to be used for **DeliveryMode.**
priority	int	Value to be used for **Priority**.
timeToLive	int64_t	Value to be used for **TimeToLive**: Length of time (milliseconds elapsed since dispatch time) that a produced message should be retained by JMS.

Return Value

None.

5.22 TopicPublisher Interface

The **TopicPublisher** interface includes the following methods:

getTopic

Syntax

```
Topic* getTopic()
```

Description

Gets the specified topic.

Return Value

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**.

Name	Type	Description
timeToLive	int64_t	Value to be used for **TimeToLive**: Length of time (milliseconds elapsed since dispatch time) that a produced message should be retained by JMS.

Return Value

None.

publish

Syntax

```
void publish(message, deliveryMode, priority, timeToLive)
```

Description

Publishes the specified message to the topic, overriding one or more default values for properties of the specified topic publisher.

Parameters

Name	Type	Description
message	BytesMessage*	Pointer to the message to be published.
deliveryMode	int	Value to be used for **DeliveryMode**.
priority	int	Value to be used for **Priority**.
timeToLive	int64_t	Value to be used for **TimeToLive**: Length of time (milliseconds elapsed since dispatch time) that a produced message should be retained by JMS.

Return Value

None.

publish

Syntax

```
void publish(message, deliveryMode, priority, timeToLive)
```

Description

Publishes the specified message to the topic, overriding one or more default values for properties of the specified topic publisher.

Parameters

Name	Type	Description
message	TextMessage*	Pointer to the message to be published.
deliveryMode	int	Value to be used for **DeliveryMode**.

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(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**.

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.

5.23 QueueSession Interface

The QueueSession Interface methods are:

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.

5.24 TopicSession Interface

The TopicSession Interface methods are:

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.

5.25 Xid Interface

The Xid Interface has the following methods:

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*

5.26 XAResource Interface

The XAResource Interface has the following methods:

- **commit** on page 231
- ****recover** on page 231
- **rollback** on page 232
- **getTransactionTimeout** on page 232
- **isSameRM** on page 233
- **prepare** on page 233
- **start** on page 234
- **end** on page 234

- **setTransactionTimeout** on page 232

Flag definitions

TMENDRSCAN	End a recovery scan
TMFAIL	Disassociates the caller and mark the transaction branch rollback-only.
TMJOIN	Caller is joining existing transaction branch.
TMNOFLAGS	Use TMNOFLAGS to indicate no flags value is selected.
TMONEPHASE	Caller is using one-phase optimization.
TMRESUME	Caller is resuming association with suspended transaction branch.
TMSTARTRSCAN	Start a recovery scan.
TMSUCCESS	Disassociate caller from transaction branch
TMSUSPEND	Caller is suspending (not ending) association with transaction branch.
XA_OK	The transaction work has been prepared normally.
XA_RDONLY	The transaction branch has been read-only and has been committed.

commit

Syntax

```
void commit(xid, onePhase)
```

Description

Commits the global transaction specified by xid.

Parameters

Name	Type	Description
xid	Xid*	A pointer to the global transaction identifier.
onePhase	STCBOOL	Flag; if true, a one-phase commit protocol is used to commit the work done on behalf of xid.

Return Value

void.

**recover

Syntax

```
Xid **recover(flag)
```

Description

This method is used during recovery to obtain the list of transaction branches that are currently in prepared or heuristically completed states.

Parameters

Name	Type	Description
flag	int	One of TMSTARTRSCAN, TMENDRSCAN, TMNOFLAGS. TMNOFLAGS must be used when no other flags are set in flags.

Return Value

Xid.

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

Return Value

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.

Return Value

None.

setTransactionTimeout

Syntax

```
STCBOOL setTransactionTimeout(seconds)
```

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

Parameters

Name	Type	Description
seconds	int	The number of seconds to time out.

Return Value

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
xares	XAResource	The resource manager instance.

Return Value

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

Return Value

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

Return Value

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 to 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

Return Value

None.

5.27 MSGSRVC_API *Lookup

The methods for the MSGSRVC_API *Lookup are:

*LookupQueueConnectionFactory

Syntax

```
*LookupQueueConnectionFactory(dllname, initString, hostname, port,
usPortOffset, iMaxRetires)
```

Description

Constructs a QueueConnectionFactory for the specified host and port using the given <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.

Return Value

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.

Return Value

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.

Return Value

Queue

*LookupTopicConnectionFactory

Syntax

```
*LookupTopicConnectionFactory(dllname, initString, hostname, port,
usPortOffset, iMaxRetires)
```

Description

Constructs a TopicConnectionFactory for the specified host and port using the given <iniString> name. You can replace this method if you are using your own naming service lookup API call.

Parameters

Name	Type	Description
dllname	const char*	dllname should always be set to MSCLIENT_DLL_NAME unless you rename the default libraries shipped with the API Kit.
initString	const char*	User defined name.
hostName	const char*	The default is DEFAULT_SERVER_NAME if you do not define a name.
port	unsigned short	The default is DEFAULT_PORT if you do not define a port.
usPortOffset	unsigned short	The default port offset is 0 if you do not define an offset.
iMaxRetries	int	The default number of maximum retries is 30 if you do not define a maximum number of retries.

Return Value

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.

Return Value

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.

Return Value

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.

Return Value

Xid

*LookupXADataSource

Syntax

```
*LookupXADataSource(const char*, const char*)
```

Description

Used to retrieve the XADataSource using a third-party 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.

Return Value

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.

Return Value

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.

Return Value

XAQueueConnectionFactory

*LookupXATopicQueueConnectionFactoryExt

Syntax

```
*LookupXATopicConnectionFactoryExt(dllname, initString, hostname,
port, usPortOffset, iMaxRetires, iInterval)
```

Description

Constructs XATopicConnectionFactoryExt for the specified host and port using the given <iniString> name. You can replace this method if you are using your own naming service lookup API call.

Parameters

Name	Type	Description
dllname	const char*	dllname should always be set to MSCLIENT_DLL_NAME unless you rename the default libraries shipped with the API Kit.
initString	const char*	User defined name.
hostName	const char*	The default is DEFAULT_SERVER_NAME if you do not define a name.
port	unsigned short	The default is DEFAULT_PORT if you do not define a port.
usPortOffset	unsigned short	The default port offset is 0 if you do not define an offset.
iMaxRetries	int	The default number of maximum retries is 30 if you do not define a maximum number of retries.
iInterval	int	The default retry interval is 100 millisecond if you do not define the retry interval.

Return Value

XATopicQueueConnectionFactory

5.28 Error Codes and Messages

The C++ API defines error codes as shown in Table 16.

Table 16 Error Codes Defined in the C++ API

Error Code	Explanation
0x80040300 = 2147746560 decimal JE_CODE_E_GENERAL	JMS exception, unspecified.
0x80040301 = 2147746561 decimal JE_CODE_E_REALLOC	A JMS exception occurred as a result of memory reallocation.
0x80040302 = 2147746562 decimal JE_CODE_E_MALLOC	A JMS exception occurred as a result of memory allocation.
0x80040303 = 2147746563 decimal JE_CODE_E_CONNECTION	A JMS exception occurred in setting up a connection.
0x80040304 = 2147746564 decimal JE_CODE_E_CREATION	A JMS exception occurred while creating a JMS object.
0x80040305 = 2147746565 decimal JE_CODE_E_CLOSED_SOCKET	A JMS exception occurred because of a closed socket.
0x80040306 = 2147746566 decimal JE_CODE_E_EOF	Processing ended because the BytesMessage or StreamMessage ended unexpectedly.
0x80040307 = 2147746567 decimal JE_CODE_E_NOTREADABLE	Processing ended because the message could not be read.

Table 16 Error Codes Defined in the C++ API

Error Code	Explanation
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.
0x80040311 = 2147746577 decimal JE_CODE_E_INVLD_SELECTOR	Processing ended because the message selector was found to be syntactically invalid.
0x80040312 = 2147746578 decimal JE_CODE_E_JMS_SECURITY	Processing was ended by JMS Security — for example, the provider rejected a name/password combination submitted by a client.
0x80040313 = 2147746579 decimal JE_CODE_E_RESOURCE_ALLOC	Processing ended because of the provider was unable to allocate resources required for the method/function.
0x80040314 = 2147746580 decimal JE_CODE_E_XA_IN_PROGRESS	Processing ended because a transaction was in progress.

Chapter 6

Working with the C and C++ Samples

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

What's in This Chapter

6.1 About the C and C++ Samples

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

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

The sample file, **eGateAPIKit_Sample.zip**, contains the **.zip** files listed in Table 17. The table describes what each **.zip** file contains.

Table 17 eGate API Kit Samples

File Name	Contents
CodeSamples.zip	The sample code files for use on Windows.
CodeSamplesUNIX.tar	The sample code files for use on UNIX operating systems.
Sample_Project.zip	A Java CAPS Project that you can import into Enterprise Designer.

6.2 Implementing the Java CAPS Projects

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

Before continuing, make sure you have downloaded the sample file as described in **"Installing the eGate API Kit" on page 19**. Implementing the sample Projects consists of the following steps:

6.2.1 Importing the Sample Projects

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

To import the sample Project into Enterprise Designer

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

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

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

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

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

6.2.2 Creating the Environment

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

6.2.3 Deploying the Projects

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

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

6.3 Configuring the Sample Environment

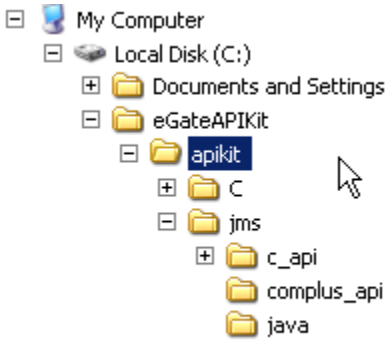
In order to compile the sample client code (or any client applications you create), you must edit the PATH variable (or the library path for UNIX) to include the path to the library files you downloaded during installation (see **"Post-Installation Instructions" on page 20**). Make sure this has been completed before performing the following steps.

In addition, the sample files must be located in a specific directory in relation to the API kit library files, as described below.

To set up the directory structure

1. Navigate to the location where you extracted **eGateAPIKIT_Sample.zip**.
2. From the extracted files, extract **CodeSamples.zip** (for Windows) or **CodeSamplesUNIX.zip** (for UNIX).
3. In the extracted files, navigate to the \apikit folder and copy the \C folder to the location where you installed the eGate API Kit at the same level as the \jms folder (see Figure 8). The \C folder contains all of the sample files.

Figure 8 Sample Directory Structure



6.4 Building the Sample C and C++ Applications

The C and C++ applications in Windows were built using Microsoft .NET 2003. A solution file is included that automatically builds all C and C++ applications, but it can only be run on .NET. A similar predefined build file (GNUmakefile) is provided to compile the applications in UNIX, which requires GNUMake. If you are using a different compiler than .NET or GNUMake, you can create a custom file to build the applications.

You can modify the sample code before compiling it if desired (to view excerpts of the sample code, see **Appendix A "C Sample Code"** and **Appendix B "C++ Sample Code"**).

To build the sample C and C++ applications in Windows

1. Navigate to the location where you copied the C folder (in **"To set up the directory structure" on page 247**).
2. Double-click the file **AllSamples.sln**.

   The project opens in .NET.
3. On the .NET toolbar, select either **Debug** or **Release** for the type of build you want to create (make sure that the **.dll** files for the appropriate type of build - Debug or Release - are entered in your PATH environment variable).
4. Click **Build** and then click **Build Solutions**.

   The executable files are created in the samples directory.

To build the sample C and C++ applications in UNIX

1. Navigate to the location where you copied the C folder (in **"To set up the directory structure" on page 247**).
2. From the command line, type the path and name of the make executable; for example:

```
/Gnu/Make/Path/make
```

The executable files are created in the samples directory.

6.5 Running the Sample C and C++ Applications

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

Table 18 lists each messaging type demonstrated in the samples along with their corresponding compiled file names.

Table 18 Class Names for C and C++ Samples

Messaging Type	Compiled C Name	Compiled C++ Name
Queue Send and Receive	queue_c	queue_cpp
Queue Requestor	queue_requestor_c	queue_requestor_cpp
Topic Publish and Subscribe	topic_c	topic_cpp
Topic Requestor	topic_requestor_c	topic_requestor_cpp
Topic Selector Publish and Subscribe	selector_c	selector_cpp
XA Publish and Subscribe	XA_c	XA_c

The commands to run the compiled code require the following flags:

- one of the following (these are not used for the requestor samples)
  - **-c** to run the application as a consumer.
  - **-u** to run the application as a producer.
- **-r** to run as requestor (used only for requestor samples).
- **-h** <host>, where <host> is the name of the server on which the Logical Host is running.
- **-p** <port>, where <port> is the IQ Manager port number of the Logical Host (18007 by default).

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

1. Open two command line windows.
2. In the first command line, run the consumer sample; for example:

```
topic_c -c -h localhost -p 18007
```

or

```
topic_cpp -c -h localhost -p 18007
```

3. In the second command line, run the producer sample; for example:

```
topic_c -u -h localhost -p 18007
```

or

```
topic_cpp -u -h localhost -p 18007
```

4 In the consumer command line, do one of the following:

- To receive additional messages, type **r** and press **Enter**.
- To exit, type **q** and press **Enter**.

To run a requestor sample application

1 Open a command line window.

2 In the command line, run the requestor sample; for example:

```
topic_requestor_c -r -h localhost -p 18007
```

or

```
topic_requestor_cpp -r -h localhost -p 18007
```

The text of the default message appears along with a comment that the message was processed.

Appendix A

C Sample Code

The C sample code is accessed from the sample files downloaded from the Documentation tab of the installer (see **"About the C and C++ Samples" on page 245** for more information). The samples were built using the compiler shown for the Windows operating system in **Table 2 on page 19**. This appendix lists the sample code provided for the types of messaging listed below.

What's in This Appendix

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,"");
```

A.1 Publish/Subscribe Messaging Using C

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

```
(20)  #include "mscapi.h"
(21)  #include <stdlib.h>
(22)  #include <stdio.h>
(23)  #include <string.h>
(24)  #ifndef WIN32
(25)  #include <unistd.h>
(26)  #endif
(27)  #if defined(WIN32)
(28)  #include "sbyn_getopt.h"
(29)  #endif
(30)
(31)  #if defined(OS400)
(32)  extern char *optarg;
(33)  #endif
(34)
(35)  #if defined(__gnu__)
(36)  #include <getopt.h>
(37)  #endif
(38)
(39)  char                    optionProducer[] = "[ -u ] [ -p port ]
(40)                          [ -h hostname ]";
(41)  char                    optionConsumer[] = "[ -c ] [ -p port ]
(42)                          [ -h hostname ]";
(43)  char                    optdescription[] = "\t-u   run as a
(44)                          producer\n\t-c run as a consumer\n\t-p
(45)                          port number\n\t-h   hostname\n";
(46)  static char             localHost[] = "localhost";
(47)  static unsigned short   susPort = 24053; /* default port number */
(48)  unsigned long           sulMessageSize = 16; /* default host name */
(49)  static char*            spHostName;
(50)  static void             subscriber();
(51)  static void             publisher();
(52)
(53)  /* Check for errors. */
(54)  static void check_error(int err, char* errBuf, int exitnow)
(55)  {
(56)       if (err){
(57)            printf("ERROR:0x%x - %s\n", err, errBuf);
(58)            if (exitnow)
(59)                 exit(1);
(60)       }
(61)  }
(62)
(63)  int main(int argc, char *argv[]) {
(64)       int           c;
(65)       char          cOption = 0;
(66)
(67)       spHostName = localHost;
(68)
(69)       while((c = getopt(argc, argv, ":p:h:P:H:cCUu")) != -1) {
(70)            switch(c){
(71)            case 'p':
(72)            case 'P':
(73)                 susPort = atoi(optarg);  /* setup the port number */
(74)                 break;
(75)            case 'h':
(76)            case 'H':
(77)                 spHostName = optarg;     /* setup the hostname */
(78)                 break;
(79)            case 'U':
(80)            case 'u':
(81)                 cOption = 'u';           /* run as a producer */
(82)                 break;
(83)            case 'c':
```

```
(84)            case 'C':
(85)                 cOption = 'c';           /* run as a consumer */
(86)                 break;
(87)            case ':':
(88)            case '?':
(89)                 printf("\nSYNOPSIS\n");
(90)                 printf("%s %s\n", argv[0], optionProducer);
(91)                 printf("%s %s\n", argv[0], optionConsumer);
(92)                 printf("%s\n", optdescription);
(93)                 exit(1);
(94)                 break;
(95)            }
(96)       }
(97)
(98)       if (cOption == 'u'){
(99)            publisher();          /* invoke producer */
(100)      } else if (cOption == 'c'){
(101)           subscriber();         /* invoke consumer */
(102)      } else {
(103)           printf("\nSYNOPSIS\n");
(104)           printf("%s %s\n", argv[0], optionProducer);
(105)           printf("%s %s\n", argv[0], optionConsumer);
(106)           printf("%s\n", optdescription);
(107)           exit(1);
(108)      }
(109) }
(110)
(111)
(112) /*
(113)  * ============================================================
(114)  * Topic Publisher
(115)  * This routine demonstrates how to publish to a topic.
(116)  * ============================================================
(117) */
(118) static void publisher() {
(119)      SBYN_TopicPublisher*    pTopicPublisher;
(120)      SBYN_Session*           pTopicSession;
(121)      SBYN_Destination*       pTopic;
(122)      SBYN_Message*           pTextMessage;
(123)      SBYN_Connection*        pTopicConnection;
(124)      SBYN_TopicConnectionFactory* pTcf;
(125)      int                     iErr;
(126)      char                    szErrBuf[256];
(127)      char                    pBuffer[] = "This is a text message";
(128)      char                    pTopicName[] = "PubSubSample";
(129)      int                     iMessagePriority = 4;
(130)      long                    iTimeToLive = 0;
(131)
(132)      /* Create a topic factory. */
(133)      pTcf = CreateTopicConnectionFactory(spHostName, susPort, &iErr,
(134)      szErrBuf);
(135)      check_error(iErr, szErrBuf, 1);
(136)      if(!pTcf) {
(137)           printf("CreateTopicConnectionFactory
(138)      failed\nError:%0X\nReason:%s\n", iErr, szErrBuf);
(139)           exit(2);
(140)      }
(141)
(142)      /* Create a topic connection. */
(143)      pTopicConnection = CreateTopicConnection(pTcf,&iErr, szErrBuf);
(144)      check_error(iErr, szErrBuf, 1);
(145)
(146)      /* Set the client ID. */
(147)      ConnectionSetClientID(pTopicConnection,
```

```
(148)      (char*)"TopicTestPublisher", &iErr, szErrBuf);
(149)      check_error(iErr, szErrBuf, 1);
(150)
(151)      /* Start the connection. */
(152)      ConnectionStart(pTopicConnection,&iErr, szErrBuf);
(153)      check_error(iErr, szErrBuf, 1);
(154)
(155)      /* Create a topic session. */
(156)      pTopicSession = ConnectionCreateTopicSession (pTopicConnection,
(157)      SBYN_TRANSACTED, SBYN_CLIENT_ACKNOWLEDGE, &iErr, szErrBuf);
(158)      check_error(iErr, szErrBuf, 1);
(159)      if(!pTopicSession) {
(160)           printf("CreateTopicSession failed\nError:%0X\nReason:%s\n",
(161)      iErr, szErrBuf);
(162)           exit(2);
(163)      }
(164)
(165)      /* Create a topic. */
(166)      pTopic = SessionCreateTopic(pTopicSession, pTopicName, &iErr,
(167)      szErrBuf);
(168)      check_error(iErr, szErrBuf, 1);
(169)
(170)      /* Create a topic publisher. */
(171)      pTopicPublisher = SessionCreatePublisher(pTopicSession, pTopic,
(172)      &iErr, szErrBuf);
(173)      check_error(iErr, szErrBuf, 1);
(174)
(175)      /* Create a text message. */
(176)      pTextMessage = SessionCreateTextMessage(pTopicSession, &iErr,
(177)      szErrBuf);
(178)      check_error(iErr, szErrBuf, 1);
(179)
(180)      /* Clear the body (payload) of the message. */
(181)      ClearBody(pTextMessage, &iErr, szErrBuf);/* set the mode to r/w */
(182)      check_error(iErr, szErrBuf, 1);
(183)
(184)      /* Copy in the text to be sent. */
(185)      SetText(pTextMessage, pBuffer, &iErr, szErrBuf);
(186)      check_error(iErr, szErrBuf, 1);
(187)
(188)      /* Set the JMSType of the message to "ASCII". */
(189)      SetJMSType(pTextMessage, (char*)"ASCII",&iErr, szErrBuf);
(190)      check_error(iErr, szErrBuf, 1);
(191)      printf("Sending Text Message: %s\n", pBuffer);
(192)
(193)      /* Publish the message. */
(194)      TopicPublisherPublish(pTopicPublisher, pTextMessage, &iErr,
(195)      szErrBuf);
(196)      check_error(iErr, szErrBuf, 1);
(197)
(198)      /* Commit the session. */
(199)      SessionCommit(pTopicSession, &iErr, szErrBuf);
(200)      check_error(iErr, szErrBuf, 1);
(201)
(202)      /* Close and clean up. */
(203)      TopicPublisherClose(pTopicPublisher, &iErr, szErrBuf);
(204)      check_error(iErr, szErrBuf, 1);
(205)      SessionClose(pTopicSession, &iErr, szErrBuf);
(206)      check_error(iErr, szErrBuf, 1);
(207)      ConnectionClose(pTopicConnection, &iErr, szErrBuf);
(208)      check_error(iErr, szErrBuf, 1);
(209)      DeleteMessage(pTextMessage, &iErr, szErrBuf);
(210)      DeleteTopicPublisher(pTopicPublisher, &iErr, szErrBuf);
(211)      DeleteSession(pTopicSession, &iErr, szErrBuf);
```

```
(212)      DeleteDestination(pTopic, &iErr, szErrBuf);
(213)      DeleteConnection(pTopicConnection, &iErr, szErrBuf);
(214)      DeleteTopicConnectionFactory(pTcf, &iErr, szErrBuf);
(215) }
(216)
(217)
(218) /*
(219)  * ========================================================
(220)  * Topic Subscriber
(221)  * This routine demonstrates how to subsribe a message from
(222)  * a topic.
(223)  * ========================================================
(224) */
(225) static void subscriber() {
(226)      SBYN_Session*           pTopicSession;
(227)      SBYN_Destination*       pTopic;
(228)      SBYN_Message*           pReceivedMessage = 0;
(229)      SBYN_TopicSubscriber*   pTopicSubscriber;
(230)      SBYN_Connection*        pTopicConnection;
(231)      SBYN_TopicConnectionFactory* pTcf;
(232)      unsigned long           ulMessageSize = 1024;
(233)      unsigned long           ulMessageCount = 10;
(234)      int                     iErr;
(235)      char                    szErrBuf[256];
(236)      char                    szUserInput[80];
(237)      char                    pTopicName[] = "eGatePubSubSample";
(238)
(239)      /* create a topic connection */
(240)      pTcf = CreateTopicConnectionFactory(spHostName, susPort, &iErr,
(241)      szErrBuf);
(242)      check_error(iErr, szErrBuf, 1);
(243)      if(!pTcf) {
(244)           printf("CreateTopicConnectionFactory
(245)      failed\nError:%0X\nReason:%s\n", iErr, szErrBuf);
(246)           exit(2);
(247)      }
(248)
(249)      /* create a topic connection */
(250)      pTopicConnection = CreateTopicConnection(pTcf,&iErr, szErrBuf);
(251)      check_error(iErr, szErrBuf, 1);
(252)
(253)      /* set client ID */
(254)      ConnectionSetClientID(pTopicConnection,
(255)      (char*)"TopicTestSubConnection",&iErr, szErrBuf);
(256)      check_error(iErr, szErrBuf, 1);
(257)
(258)      /* start connection */
(259)      ConnectionStart(pTopicConnection,&iErr, szErrBuf);
(260)      check_error(iErr, szErrBuf, 1);
(261)
(262)      /* create a topic session */
(263)      pTopicSession = ConnectionCreateTopicSession (pTopicConnection,
(264)      SBYN_TRANSACTED, SBYN_CLIENT_ACKNOWLEDGE, &iErr, szErrBuf);
(265)      check_error(iErr, szErrBuf, 1);
(266)      if(!pTopicSession) {
(267)           printf("CreateTopicSession failed\nError:%0X\nReason:%s\n",
(268)      iErr, szErrBuf);
(269)           exit(2);
(270)      }
(271)
(272)      /* create a topic */
(273)      pTopic = SessionCreateTopic(pTopicSession, pTopicName, &iErr,
(274)      szErrBuf);
(275)      check_error(iErr, szErrBuf, 1);
```

```
(276)
(277)      /* create a subscriber */
(278)      pTopicSubscriber = SessionCreateDurableSubscriber(pTopicSession,
(279)      pTopic, (char*)"TopicTestSubscriber",&iErr, szErrBuf);
(280)      check_error(iErr, szErrBuf, 1);
(281)
(282)      printf("Waiting for message ... \n");
(283)      do {
(284)           /* waiting for incoming messages */
(285)           pReceivedMessage = TopicSubscriberReceive(pTopicSubscriber,
(286)      &iErr, szErrBuf);
(287)           check_error(iErr, szErrBuf, 1);
(288)           if (pReceivedMessage->type == SBYN_MESSAGE_TYPE_TEXT){
(289)                char *pReturnedBuf;
(290)                SBYN_WString *pWstr;
(291)                char* pMessageType;
(292)                /* retrive the JMS message type */
(293)                pWstr = GetJMSType(pReceivedMessage, &iErr, szErrBuf);
(294)                check_error(iErr, szErrBuf, 1);
(295)                pMessageType = WStringToChar(pWstr);
(296)                check_error(iErr, szErrBuf, 1);
(297)
(298)                /* retrive the text from message */
(299)              pReturnedBuf = GetText(pReceivedMessage, &iErr, szErrBuf);
(300)                printf("Received text message (JMSType:%s):  %s\n",
(301)      pMessageType, pReturnedBuf);
(302)                free(pReturnedBuf);
(303)                DeleteWString(pWstr);
(304)                free((void*)pMessageType);
(305)            SessionCommit(pTopicSession, &iErr, szErrBuf);
(306)           }
(307)           printf("Enter 'r' for receiving more message, 'q' for
(308)      exit\n");
(309)           scanf("%s", szUserInput);
(310)      } while ( szUserInput[0] != 'q' && szUserInput[0] != 'Q' );
(311)
(312)      check_error(iErr, szErrBuf, 1);
(313)
(314)      /* close subscriber, session ... */
(315)      TopicSubscriberClose(pTopicSubscriber, &iErr, szErrBuf);
(316)      SessionClose(pTopicSession, &iErr, szErrBuf);
(317)      ConnectionClose(pTopicConnection, &iErr, szErrBuf);
(318)
(319)      /* delete objects */
(320)      DeleteMessage(pReceivedMessage, &iErr, szErrBuf);
(321)      DeleteDestination(pTopic, &iErr, szErrBuf);
(322)      DeleteTopicSubscriber(pTopicSubscriber, &iErr, szErrBuf);
(323)      DeleteSession(pTopicSession, &iErr, szErrBuf);
(324)      DeleteConnection(pTopicConnection, &iErr, szErrBuf);
(325)      DeleteTopicConnectionFactory(pTcf, &iErr, szErrBuf);
(326) }
```

A.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) }
```

A.3 Request-Reply Messaging Using C

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

```
(22)  #include "mscapi.h"
(23)  #include <stdlib.h>
(24)  #include <stdio.h>
(25)  #include <string.h>
(26)
(27)  #ifndef WIN32
(28)  #include <unistd.h>
(29)  #endif
(30)  #if defined(WIN32)
(31)  #include <sbyn_getopt.h>
(32)  #endif
(33)
(34)  #if defined(OS400)
(35)  extern char *optarg;
(36)  #endif
(37)
(38)  #if defined(__gnu__)
(39)  #include <getopt.h>
(40)  #endif
(41)
(42)  static void requestor();
(43)
(44)  /* Routine for checking errors.*/
(45)  static void check_error(int err, char* errBuf, int exitnow)
(46)  {
(47)       if (err){
(48)            printf("ERROR:0x%x - %s\n", err, errBuf);
(49)            if (exitnow)
(50)                 exit(1);
(51)       }
(52)  }
(53)
(54)  char                    optionRequestor[] = "[ -r ] [ -p port ]
(55)                          [ -h hostname ]";
(56)  char                    optdescription[] = "\t-r run as a
(57)                          requestor\n\t-p port number\n\t-h
(58)                          hostname\n";
(59)  char*                   spHostName;
(60)  char                    localHost[] = "localhost";
(61)  static unsigned short   susPort = 24053;
(62)  int                     iErr;
(63)  char                    szErrBuf[256];
(64)  char                    pQueueName[] = "QueueRequestorSample";
(65)
(66)  int main(int argc, char *argv[]) {
(67)       int           c;
(68)       char          cOption = 0;
(69)
(70)       spHostName = localHost;
(71)
(72)       while((c = getopt(argc, argv, ":p:h:P:H:rR")) != -1) {
(73)            switch(c){
(74)            case 'p':
(75)            case 'P':
(76)                 susPort = atoi(optarg);  /* setup the port number */
(77)                 break;
(78)            case 'h':
(79)            case 'H':
(80)                 spHostName = optarg;     /* setup the hostname */
(81)                 break;
(82)            case 'R':
(83)            case 'r':
(84)                 cOption = 'r';           /* run as a requestor */
(85)                 break;
```

```
(86)            case ':':
(87)            case '?':
(88)                 printf("\nSYNOPSIS\n");
(89)                 printf("%s %s\n", argv[0], optionRequestor);
(90)                 printf("%s\n", optdescription);
(91)                 exit(1);
(92)             }
(93)       }
(94)
(95)       if (cOption == 'r'){
(96)            requestor();/* invoke requestor */
(97)       } else {
(98)            printf("\nSYNOPSIS\n");
(99)            printf("%s %s\n", argv[0], optionRequestor);
(100)           printf("%s\n", optdescription);
(101)           exit(1);
(102)      }
(103) }
(104)
(105)
(106)
(107) /*
(108)  * ============================================================
(109)  * Queue Requestor
(110)  *   This routine demonstrates how to do request/reply.
(111)  * ============================================================
(112) */
(113) void requestor(){
(114)      SBYN_QueueConnectionFactory *pQcf     = NULL;
(115)      SBYN_Connection *pQueueConnection     = NULL;
(116)      SBYN_Session *pQueueSession           = NULL;
(117)      SBYN_Destination *pQueue              = NULL;
(118)      SBYN_QueueSender *pQueueSender        = NULL;
(119)      SBYN_Message *textMessage             = NULL;
(120)      SBYN_Message *pReplyMessage           = NULL;
(121)      SBYN_QueueRequestor *pQueueRequestor  = 0;
(122)      char pBuffer[] = "This is a text message";
(123)
(124)      /* Create a queue connection factory */
(125)      pQcf = CreateQueueConnectionFactory(spHostName, susPort, &iErr,
      szErrBuf);
(126)      check_error(iErr, szErrBuf, 1);
(127)      if(!pQcf) {
(128)           printf("CreateQueueConnectionFactory
      failed\nError:%0X\nReason:%s\n", iErr, szErrBuf);
(129)           exit(2);
(130)      }
(131)
(132)      /* Create a queue connection */
(133)      pQueueConnection = CreateQueueConnection(pQcf,&iErr, szErrBuf);
(134)      check_error(iErr, szErrBuf, 1);
(135)
(136)      /* Set the client ID */
(137)      ConnectionSetClientID(pQueueConnection, (char*)"REQUESTOR",&iErr,
      szErrBuf);
(138)      check_error(iErr, szErrBuf, 1);
(139)
(140)      /* Start the connection */
(141)      ConnectionStart(pQueueConnection,&iErr, szErrBuf);
(142)      check_error(iErr, szErrBuf, 1);
(143)
(144)      /* Create a queue session */
(145)      pQueueSession = ConnectionCreateQueueSession (pQueueConnection,
      SBYN_NON_TRANSACTED, SBYN_CLIENT_ACKNOWLEDGE, &iErr, szErrBuf);
```

```
(146)      check_error(iErr, szErrBuf, 1);
(147)      if(!pQueueSession) {
(148)           printf("CreateTopicSession failed\nError:%0X\nReason:%s\n",
     iErr, szErrBuf);
(149)           exit(2);
(150)      }
(151)
(152)      /* Create a queue */
(153)      pQueue = SessionCreateQueue(pQueueSession, pQueueName, &iErr,
     szErrBuf);
(154)      check_error(iErr, szErrBuf, 1);
(155)
(156)      /* Create a queue requestor */
(157)      pQueueRequestor = CreateQueueRequestor(pQueueSession,pQueue,
     &iErr, szErrBuf);
(158)      check_error(iErr, szErrBuf, 1);
(159)
(160)      /* Create a text message and make a request */
(161)      textMessage = SessionCreateTextMessage(pQueueSession, &iErr,
     szErrBuf);
(162)      check_error(iErr, szErrBuf, 1);
(163)
(164)      /* set the mode to r/w */
(165)      ClearBody(textMessage, &iErr, szErrBuf);
(166)      check_error(iErr, szErrBuf, 1);
(167)
(168)      /* Copy in the text to be sent. */
(169)      SetText(textMessage, pBuffer, &iErr, szErrBuf);
(170)      check_error(iErr, szErrBuf, 1);
(171)      printf("Sending Message: %s\n", pBuffer);
(172)
(173)      /* Set ReplyTo destination */
(174)      SetJMSReplyTo(textMessage, pQueue, &iErr, szErrBuf);
(175)      check_error(iErr, szErrBuf, 1);
(176)
(177)      /* Make a request and wait for a reply */
(178)      pReplyMessage = QueueRequestorRequestTimeOut(pQueueRequestor,
     textMessage, 100000, &iErr, szErrBuf);
(179)      check_error(iErr, szErrBuf, 1);
(180)
(181)      /* Extract the message type */
(182)      if (GetMessageType(pReplyMessage, &iErr, szErrBuf) ==
     SBYN_MESSAGE_TYPE_TEXT){
(183)           char *rtbuf;
(184)   check_error(iErr, szErrBuf, 1);
(185)   /* Extract the text */
(186)           rtbuf = GetText(pReplyMessage, &iErr, szErrBuf);
(187)   check_error(iErr, szErrBuf, 1);
(188)           printf("Received message:  %s\n", rtbuf);
(189)           free(rtbuf);
(190)      }
(191)      DeleteMessage(pReplyMessage, &iErr, szErrBuf);
(192)      check_error(iErr, szErrBuf, 1);
(193)
(194)      /* now close the connections */
(195)      SessionClose(pQueueSession, &iErr, szErrBuf);
(196)      check_error(iErr, szErrBuf, 1);
(197)      ConnectionStop(pQueueConnection, &iErr, szErrBuf);
(198)      check_error(iErr, szErrBuf, 1);
(199)      ConnectionClose(pQueueConnection, &iErr, szErrBuf);
(200)      check_error(iErr, szErrBuf, 1);
(201)
(202)      /* delete the objects */
(203)      DeleteMessage(textMessage, &iErr, szErrBuf);
```

```
(204)      DeleteQueueSender(pQueueSender, &iErr, szErrBuf);
(205)      DeleteDestination(pQueue, &iErr, szErrBuf);
(206)      DeleteSession(pQueueSession, &iErr, szErrBuf);
(207)      DeleteConnection(pQueueConnection, &iErr, szErrBuf);
(208)      DeleteQueueConnectionFactory(pQcf, &iErr, szErrBuf);
(209) }
(210)
(211)
(212)
(213) void receiver(){
(214)      SBYN_QueueConnectionFactory* pQcf = NULL;
(215)      SBYN_Connection* pQueueConnection = NULL;
(216)      SBYN_Session* pQueueSession = NULL;
(217)      SBYN_Destination* pQueue = NULL;
(218)      SBYN_Destination* pReplyToQueue = NULL;
(219)      SBYN_QueueReceiver* pQueueReceiver = NULL;
(220)      SBYN_QueueSender* pReplyQueueSender = NULL;
(221)      SBYN_Message *pMessage = NULL;
(222)      char szUserInput[80];
(223)
(224)      printf("Queue name is %s\n", pQueueName);
(225)      pQcf = CreateQueueConnectionFactory(spHostName, susPort, &iErr,
      szErrBuf);
(226)      if(!pQcf) {
(227)           printf("CreateQueueConnectionFactory
      failed\nError:%0X\nReason:%s\n", iErr, szErrBuf);
(228)           exit(2);
(229)      }
(230)
(231)      pQueueConnection = CreateQueueConnection(pQcf,&iErr, szErrBuf);
(232)      ConnectionSetClientID(pQueueConnection, (char*)"RECEIVER",&iErr,
      szErrBuf);
(233)      ConnectionStart(pQueueConnection,&iErr, szErrBuf);
(234)      pQueueSession = ConnectionCreateQueueSession (pQueueConnection,
      SBYN_TRANSACTED, SBYN_CLIENT_ACKNOWLEDGE, &iErr, szErrBuf);
(235)      if(!pQueueSession) {
(236)           printf("CreateTopicSession failed\nError:%0X\nReason:%s\n",
      iErr, szErrBuf);
(237)           exit(2);
(238)      }
(239)      pQueue = SessionCreateQueue(pQueueSession, pQueueName, &iErr,
      szErrBuf);
(240)      pQueueReceiver = SessionCreateReceiver(pQueueSession, pQueue,
      &iErr, szErrBuf);
(241)
(242)      do {
(243)           pMessage = QueueReceiverReceive(pQueueReceiver, &iErr,
      szErrBuf);
(244)           if (pMessage->type == SBYN_MESSAGE_TYPE_TEXT){
(245)                char *rtbuf;
(246)                rtbuf = GetText(pMessage, &iErr, szErrBuf);
(247)                printf("Reading message:  %s\n", rtbuf);
(248)                free(rtbuf);
(249)           } else {
(250)                printf("Error: Received invalid message format\n");
(251)           }
(252)           pReplyToQueue = GetJMSReplyTo(pMessage, &iErr, szErrBuf);
(253)           pReplyQueueSender = SessionCreateSender(pQueueSession,
      pReplyToQueue, &iErr, szErrBuf);
(254)           ClearBody(pMessage, &iErr, szErrBuf);
(255)           SetText(pMessage, (char*)"This is reply message", &iErr,
      szErrBuf);
(256)          QueueSenderSend(pReplyQueueSender, pMessage, &iErr, szErrBuf);
(257)           SessionCommit(pQueueSession, &iErr, szErrBuf);
```

```
(258)           printf("Enter 'r' for receiving more message, 'q' for
     exit\n");
(259)           scanf("%s", szUserInput);
(260)       } while ( szUserInput[0] != 'q' && szUserInput[0] != 'Q' );
(261)
(262)       /* now close the connections */
(263)       QueueReceiverClose(pQueueReceiver, &iErr, szErrBuf);
(264)       SessionClose(pQueueSession, &iErr, szErrBuf);
(265)       ConnectionStop(pQueueConnection, &iErr, szErrBuf);
(266)       ConnectionClose(pQueueConnection, &iErr, szErrBuf);
(267)
(268)       /* delete the objects */
(269)       DeleteMessage(pMessage, &iErr, szErrBuf);
(270)       DeleteQueueReceiver(pQueueReceiver, &iErr, szErrBuf);
(271)       DeleteDestination(pQueue, &iErr, szErrBuf);
(272)       DeleteSession(pQueueSession, &iErr, szErrBuf);
(273)       DeleteConnection(pQueueConnection, &iErr, szErrBuf);
(274)       DeleteQueueConnectionFactory(pQcf, &iErr, szErrBuf);
(275) }
```

A.4 Message Selector Using C

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

```
(37)
(38)  #if defined(__gnu__)
(39)  #include <getopt.h>
(40)  #endif
(41)
(42)  char                    optionProducer[] = "[ -u ] [ -p port ]
(43)                          [ -h hostname ]";
(44)  char                    optionConsumer[] = "[ -c ] [ -p port ]
(45)                          [ -h hostname ]";
(46)  char                    optdescription[] = "\t-u run as a
(47)                          producer\n\t-c run as a consumer\n\t-p
(48)                          port number\n\t-h hostname\n";
(49)  static char             localHost[] = "localhost";
(50)  static unsigned short   susPort = 24053; /* default port number */
(51)  unsigned long           sulMessageSize = 16; /* default host name */
(52)  static char*            spHostName;
(53)  static char             PROP_NAME[] = "property";
(54)  int                     iErr;
(55)  char                    szErrBuf[256];
(56)
(57)  static void selector_publisher();
(58)  static void selector_subscriber();
(59)
(60)  /* Check for errors. */
(61)  static void check_error(int err, char* errBuf, int exitnow)
(62)  {
(63)       if (err){
(64)            printf("ERROR:0x%x - %s\n", err, errBuf);
(65)            if (exitnow)
(66)                 exit(1);
(67)       }
(68)  }
(69)
(70)
(71)  int main(int argc, char *argv[]) {
(72)       int           c;
(73)       char          cOption = 0;
(74)
(75)       spHostName = localHost;
(76)
(77)       while((c = getopt(argc, argv, ":p:h:P:H:cCUu")) != -1) {
(78)            switch(c){
(79)            case 'p':
(80)            case 'P':
(81)                 susPort = atoi(optarg);  /* setup the port number */
(82)                 break;
(83)            case 'h':
(84)            case 'H':
(85)                 spHostName = optarg;     /* setup the hostname */
(86)                 break;
(87)            case 'U':
(88)            case 'u':
(89)                 cOption = 'u';           /* run as a producer */
(90)                 break;
(91)            case 'c':
(92)            case 'C':
(93)                 cOption = 'c';           /* run as a consumer */
(94)                 break;
(95)            case ':':
(96)            case '?':
(97)                 printf("\nSYNOPSIS\n");
(98)                 printf("%s %s\n", argv[0], optionProducer);
(99)                 printf("%s %s\n", argv[0], optionConsumer);
(100)                printf("%s\n", optdescription);
```

```
(101)                exit(1);
(102)                break;
(103)           }
(104)      }
(105)
(106)      if (cOption == 'u'){
(107)           selector_publisher();          /* invoke producer */
(108)      } else if (cOption == 'c'){
(109)           selector_subscriber();         /* invoke consumer */
(110)      } else {
(111)           printf("\nSYNOPSIS\n");
(112)           printf("%s %s\n", argv[0], optionProducer);
(113)           printf("%s %s\n", argv[0], optionConsumer);
(114)           printf("%s\n", optdescription);
(115)           exit(1);
(116)      }
(117) }
(118)
(119) static void selector_publisher(){
(120)      SBYN_TopicConnectionFactory*        pTcf;
(121)      SBYN_Connection*                    pTopicConnection = NULL;
(122)      SBYN_Session*                       pTopicSession = NULL;
(123)      SBYN_Destination*                   pTopic = NULL;
(124)      SBYN_TopicPublisher*                pTopicPublisher = NULL;
(125)      int                                 ii;
(126)      SBYN_Message                        msglist[10];
(127)      static char                         TOPIC_NAME[] = "Selector";
(128)
(129)      /* Create a topic factory. */
(130)      pTcf = CreateTopicConnectionFactory(spHostName, susPort, &iErr,
      szErrBuf);
(131)      if(!pTcf) {
(132)           printf("CreateTopicConnectionFactory
      failed\nError:%0X\nReason:%s\n", iErr, szErrBuf);
(133)           exit(2);
(134)      }
(135)
(136)      /* Create a topic connection. */
(137)      pTopicConnection = CreateTopicConnection(pTcf,&iErr, szErrBuf);
(138)      check_error(iErr, szErrBuf, 1);
(139)
(140)      /* Set the client ID. */
(141)      ConnectionSetClientID(pTopicConnection, (char*)"Publisher",&iErr,
      szErrBuf);
(142)      check_error(iErr, szErrBuf, 1);
(143)
(144)      /* Start the connection. */
(145)      ConnectionStart(pTopicConnection, &iErr, szErrBuf);
(146)      check_error(iErr, szErrBuf, 1);
(147)
(148)      /* Create a topic session. */
(149)      pTopicSession = ConnectionCreateTopicSession (pTopicConnection,
      SBYN_TRANSACTED, SBYN_CLIENT_ACKNOWLEDGE, &iErr, szErrBuf);
(150)      check_error(iErr, szErrBuf, 1);
(151)      if(!pTopicSession) {
(152)           printf("CreateTopicSession failed\nError:%0X\nReason:%s\n",
      iErr, szErrBuf);
(153)           exit(2);
(154)      }
(155)
(156)      /* Create a topic. */
(157)      pTopic = SessionCreateTopic(pTopicSession, TOPIC_NAME, &iErr,
      szErrBuf);
(158)      check_error(iErr, szErrBuf, 1);
```

```
(159)
(160)      /* Create a topic publisher. */
(161)      pTopicPublisher = SessionCreatePublisher(pTopicSession, pTopic,
      &iErr, szErrBuf);
(162)      check_error(iErr, szErrBuf, 1);
(163)
(164)      /* Set delivery mode as persistent */
(165)      TopicPublisherSetDeliveryMode(pTopicPublisher, SBYN_PERSISTENT,
      &iErr, szErrBuf);
(166)      check_error(iErr, szErrBuf, 1);
(167)
(168)      /* publish 10 messages to the topic */
(169)      for(ii=0; ii<10 ;ii++){
(170)           int index;
(171)           char buf[80];
(172)           /* Create a text message. */
(173)          msglist[ii].message = SessionCreateTextMessage(pTopicSession,
      &iErr, szErrBuf);
(174)   /* Clear the body (payload) of the message. */
(175)           ClearBody((SBYN_Message*)msglist[ii].message, &iErr,
      szErrBuf);
(176)           check_error(iErr, szErrBuf, 1);
(177)           msglist[ii].type = SBYN_MESSAGE_TYPE_TEXT;
(178)           index = ii%10;
(179)           sprintf(buf, "%d", index);
(180)           /* Set the string property */
(181)           SetStringProperty((SBYN_Message*)msglist[ii].message,
      PROP_NAME, buf, &iErr, szErrBuf);
(182)           check_error(iErr, szErrBuf, 1);
(183)           /* Copy in the text to be sent. */
(184)          SetText((SBYN_Message*)msglist[ii].message, (char*)"This is a
      text message", &iErr, szErrBuf);
(185)           check_error(iErr, szErrBuf, 1);
(186)           /* Publish the message. */
(187)           TopicPublisherPublish(pTopicPublisher,
      (SBYN_Message*)msglist[ii].message, &iErr, szErrBuf);
(188)           check_error(iErr, szErrBuf, 1);
(189)           printf("... Published 1 message with property %s = %d\n",
      PROP_NAME, ii);
(190)      }
(191)      /* Commit the session. */
(192)      SessionCommit(pTopicSession,  &iErr, szErrBuf);
(193)      check_error(iErr, szErrBuf, 1);
(194)
(195)      /* close and delete objects */
(196)      TopicPublisherClose(pTopicPublisher, &iErr, szErrBuf);
(197)      check_error(iErr, szErrBuf, 1);
(198)      SessionClose(pTopicSession, &iErr, szErrBuf);
(199)      check_error(iErr, szErrBuf, 1);
(200)      ConnectionClose(pTopicConnection, &iErr, szErrBuf);
(201)      check_error(iErr, szErrBuf, 1);
(202)      DeleteTopicPublisher(pTopicPublisher, &iErr, szErrBuf);
(203)      DeleteSession(pTopicSession, &iErr, szErrBuf);
(204)      DeleteConnection(pTopicConnection, &iErr, szErrBuf);
(205)      DeleteTopicConnectionFactory(pTcf, &iErr, szErrBuf);
(206) }
(207)
(208)
(209) static void selector_subscriber(){
(210)      SBYN_TopicConnectionFactory *pTcf;
(211)      SBYN_Connection *pTopicConnection = 0;
(212)      SBYN_Session *pTopicSession = 0;
(213)      SBYN_Destination *topic = 0;
(214)      SBYN_TopicSubscriber *pTopicSubscriber = 0;
```

```
(215)      SBYN_Message *pMessage = 0;
(216)      char selectorString[80];
(217)      char selectorSubscriberName[80];
(218)      int selector = 7;
(219)      char* selectorName;
(220)      static char TOPIC_NAME[] = "eGateSelector";
(221)
(222)
(223)      /* create a topic connection */
(224)      pTcf = CreateTopicConnectionFactory(spHostName, susPort, &iErr,
     szErrBuf);
(225)      check_error(iErr, szErrBuf, 1);
(226)      if(!pTcf) {
(227)           printf("CreateTopicConnectionFactory
     failed\nError:%0X\nReason:%s\n", iErr, szErrBuf);
(228)           exit(2);
(229)      }
(230)
(231)      /* create a topic connection */
(232)      pTopicConnection = CreateTopicConnection(pTcf,&iErr, szErrBuf);
(233)      check_error(iErr, szErrBuf, 1);
(234)
(235)      /* set client ID */
(236)     ConnectionSetClientID(pTopicConnection, (char*)"Publisher", &iErr,
     szErrBuf);
(237)      check_error(iErr, szErrBuf, 1);
(238)
(239)      /* start connection */
(240)      ConnectionStart(pTopicConnection,&iErr, szErrBuf);
(241)      check_error(iErr, szErrBuf, 1);
(242)
(243)      /* create a topic session */
(244)      pTopicSession = ConnectionCreateTopicSession (pTopicConnection,
     SBYN_TRANSACTED, SBYN_CLIENT_ACKNOWLEDGE, &iErr, szErrBuf);
(245)      check_error(iErr, szErrBuf, 1);
(246)      if(!pTopicSession) {
(247)           printf("CreateTopicSession failed\nError:%0X\nReason:%s\n",
     iErr, szErrBuf);
(248)           exit(2);
(249)      }
(250)
(251)      /* create a topic */
(252)      topic = SessionCreateTopic(pTopicSession, TOPIC_NAME, &iErr,
     szErrBuf);
(253)      check_error(iErr, szErrBuf, 1);
(254)
(255)      /* create subscriber  with selector*/
(256)      sprintf(selectorString, "%s = '%d'", PROP_NAME, selector);
(257)      selectorString[strlen(selectorString)] = '\0';
(258)      sprintf(selectorSubscriberName, "SelectorSubscriber%d", selector);
(259)      pTopicSubscriber =
     SessionCreateDurableSubscriberMessageSelector(pTopicSession, topic,
     selectorSubscriberName, selectorString, 0, &iErr, szErrBuf);
(260)      check_error(iErr, szErrBuf, 1);
(261)
(262)      /* Get message using selector */
(263)     selectorName = TopicSubscriberGetMessageSelector(pTopicSubscriber,
     &iErr, szErrBuf);
(264)      check_error(iErr, szErrBuf, 1);
(265)      printf("using selector: %s\n", selectorName);
(266)      for (pMessage = TopicSubscriberReceive(pTopicSubscriber, &iErr,
     szErrBuf);
(267)           pMessage != 0;
```

```
(268)           pMessage = TopicSubscriberReceiveTimeOut(pTopicSubscriber,
     1000, &iErr, szErrBuf))
(269)      {
(270)           char* property = WStringToChar(GetStringProperty(pMessage,
     PROP_NAME, &iErr, szErrBuf));
(271)           printf("Received 1 message with %s = %s\n", PROP_NAME,
     property);
(272)      }
(273)      check_error(iErr, szErrBuf, 1);
(274)
(275)      /* Session commit */
(276)      SessionCommit(pTopicSession, &iErr, szErrBuf);
(277)      check_error(iErr, szErrBuf, 1);
(278)
(279)      /* close and delete objects */
(280)      TopicSubscriberClose(pTopicSubscriber, &iErr, szErrBuf);
(281)      check_error(iErr, szErrBuf, 1);
(282)      SessionClose(pTopicSession, &iErr, szErrBuf);
(283)      check_error(iErr, szErrBuf, 1);
(284)      ConnectionClose(pTopicConnection, &iErr, szErrBuf);
(285)      check_error(iErr, szErrBuf, 1);
(286)
(287)      DeleteTopicSubscriber(pTopicSubscriber, &iErr, szErrBuf);
(288)      DeleteSession(pTopicSession, &iErr, szErrBuf);
(289)      DeleteConnection(pTopicConnection, &iErr, szErrBuf);
(290)      DeleteTopicConnectionFactory(pTcf, &iErr, szErrBuf);
(291) }
```

A.5 XA Publish/Subscribe Messaging Using C

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

```
(32)  #if defined(OS400)
(33)  extern char *optarg;
(34)  #endif
(35)
(36)  #if defined(__gnu__)
(37)  #include <getopt.h>
(38)  #endif
(39)
(40)
(41)  char                    optionProducer[] = "[ -u ] [ -p port ]
(42)                          [ -h hostname ]";
(43)  char                    optionConsumer[] = "[ -c ] [ -p port ]
(44)                          [ -h hostname ]";
(45)  char                    optdescription[] = "\t-u  run as a
(46)                          producer\n\t-c run as a consumer\n\t-p port
(47)                          number\n\t-h  hostname\n";
(48)  static char             localHost[] = "localhost";
(49)  static unsigned short   susPort = 24053; /* default port number */
(50)  unsigned long           sulMessageSize = 16; /* default host name */
(51)  static char*            spHostName;
(52)  static int              iErr;
(53)  static char             szErrBuf[256];
(54)  static int              iNumMessages = 10;
(55)  static char             szText[] = "This is a text message";
(56)
(57)  static void XATopicPub();
(58)  static void XATopicSub();
(59)
(60)  #define MSCLIENT_DLL_NAME "stc_msclient.dll"
(61)
(62)  /* Check for errors. */
(63)  static void check_error(int err, char* errBuf, int exitnow)
(64)  {
(65)       if (err){
(66)            printf("ERROR:0x%x - %s\n", err, errBuf);
(67)            if (exitnow)
(68)                 exit(1);
(69)       }
(70)  }
(71)
(72)
(73)  int main(int argc, char *argv[]) {
(74)       int           c;
(75)       char          cOption = 0;
(76)
(77)       spHostName = localHost;
(78)
(79)       while((c = getopt(argc, argv, ":p:h:P:H:cCUu")) != -1) {
(80)            switch(c){
(81)            case 'p':
(82)            case 'P':
(83)                 susPort = atoi(optarg);  /* setup the port number */
(84)                 break;
(85)            case 'h':
(86)            case 'H':
(87)                 spHostName = optarg;     /* setup the hostname */
(88)                 break;
(89)            case 'U':
(90)            case 'u':
(91)                 cOption = 'u';           /* run as a producer */
(92)                 break;
(93)            case 'c':
(94)            case 'C':
(95)                 cOption = 'c';           /* run as a consumer */
```

```
(96)                 break;
(97)            case ':':
(98)            case '?':
(99)                 printf("\nSYNOPSIS\n");
(100)                printf("%s %s\n", argv[0], optionProducer);
(101)                printf("%s %s\n", argv[0], optionConsumer);
(102)                printf("%s\n", optdescription);
(103)                exit(1);
(104)                break;
(105)           }
(106)     }
(107)
(108)     if (cOption == 'u'){
(109)          XATopicPub();          /* invoke producer */
(110)     } else if (cOption == 'c'){
(111)          XATopicSub();         /* invoke consumer */
(112)     } else {
(113)          printf("\nSYNOPSIS\n");
(114)          printf("%s %s\n", argv[0], optionProducer);
(115)          printf("%s %s\n", argv[0], optionConsumer);
(116)          printf("%s\n", optdescription);
(117)          exit(1);
(118)     }
(119) }
(120)
(121) /*
(122)  * ============================================================
(123)  * Publish Message
(124)  * This routine publishes iNumMessages to the topic
(125)  * ============================================================
(126) */
(127) static void PublishMessage(SBYN_TopicPublisher* pPublisher,
(128) SBYN_Message* pMessage, int iNumMessages)
(129) {
(130)     int ii;
(131)     for ( ii = 0; ii < iNumMessages; ii++){
(132)          SetIntProperty(pMessage, (char*)"Sequence", ii, &iErr,
(133)          szErrBuf);
(134)          check_error(iErr, szErrBuf, 1);
(135)          printf("Sending Message: Sequence number %d\n", ii);
(136)          TopicPublisherPublish(pPublisher, pMessage, &iErr, szErrBuf);
(137)          check_error(iErr, szErrBuf, 1);
(138)     }
(139) }
(140)
(141)
(142) /*
(143)  * ============================================================
(144)  * Receive Message
(145)  * This routine block on receiving message for maximum iWait
(146)  *   seconds before return.
(147)  * ============================================================
(148) */
(149) static int SubscriberReceiveMessage(SBYN_TopicSubscriber* pSub)
(150) {
(151)     int iMsgCount = 0;
(152)     SBYN_Message* pRMsg = 0;
(153)     char szUserInput[8];
(154)     printf("Waiting for message ... \n");
(155)     do {
(156)          pRMsg = TopicSubscriberReceive(pSub, &iErr, szErrBuf);
(157)          printf("Received Message %d\n", iMsgCount);
(158)          check_error(iErr, szErrBuf, 1);
(159)          iMsgCount++;
```

```
(160)           if (iMsgCount >= iNumMessages){
(161)                printf("Enter 'r' for receiving more message, 'q' for
(162)                exit\n");
(163)                scanf("%s", szUserInput);
(164)                iMsgCount = 0;
(165)           }
(166)
(167)      } while(szUserInput[0] != 'q' && szUserInput[0] != 'Q');
(168)      return iMsgCount;
(169) }
(170)
(171) /*
(172)  * ============================================================
(173)  * Topic Publisher
(174)  * This routine demonstrates how to publish to a topic.
(175)  * ============================================================
(176) */
(177) void XATopicPub()
(178) {
(179)      SBYN_XATopicConnectionFactory* pXATcf = 0;
(180)      SBYN_Connection*               pConnection = 0;
(181)      SBYN_Session*                  pXATopicSession = 0;
(182)      SBYN_Session*                  pTopicSession = 0;
(183)      SBYN_Destination*              pTopic = 0;
(184)      SBYN_XAResource*               pXATopicResource;
(185)      SBYN_XAResource*               pXATopicResourceTmp;
(186)      SBYN_TopicPublisher*           pTopicPublisher;
(187)      SBYN_Message*                  pMessage;
(188)      SBYN_Xid*                      pXid;
(189)      char                           pTopicName[] = "XAPubSubSample";
(190)
(191)      /* create XA connection factory */
(192)     pXATcf = CreateXATopicConnectionFactory(spHostName, susPort, &iErr,
(193)      szErrBuf);
(194)      check_error(iErr, szErrBuf, 1);
(195)
(196)      /* create XA connection */
(197)      pConnection = CreateXATopicConnection(pXATcf, &iErr, szErrBuf);
(198)      check_error(iErr, szErrBuf, 1);
(199)
(200)      /* set client ID */
(201)      ConnectionSetClientID(pConnection, (char*)"eGate{7E527692-770A-
(202)      11D5-B139-935EB6E85DBD}", &iErr, szErrBuf);
(203)      check_error(iErr, szErrBuf, 1);
(204)
(205)      /* create XA session */
(206)      pXATopicSession = XAConnectionCreateXATopicSession(pConnection,
(207)      &iErr, szErrBuf);
(208)      check_error(iErr, szErrBuf, 1);
(209)
(210)      /* get session */
(211)      pTopicSession = XASessionGetTopicSession(pXATopicSession, &iErr,
(212)      szErrBuf);
(213)      check_error(iErr, szErrBuf, 1);
(214)
(215)      /* get XA resource */
(216)      pXATopicResource = XASessionGetXAResource(pXATopicSession, &iErr,
(217)      szErrBuf);
(218)      check_error(iErr, szErrBuf, 1);
(219)
(220)      /* get XA resource */
(221)      pXATopicResourceTmp = XASessionGetXAResource(pXATopicSession,
(222)      &iErr, szErrBuf);
(223)      check_error(iErr, szErrBuf, 1);
```

```
(224)
(225)       /* create a Topic */
(226)       pTopic = SessionCreateTopic(pTopicSession, pTopicName, &iErr,
(227)       szErrBuf);
(228)       check_error(iErr, szErrBuf, 1);
(229)
(230)       /* create a publisher */
(231)       pTopicPublisher = SessionCreatePublisher(pTopicSession, pTopic,
(232)       &iErr, szErrBuf);
(233)       check_error(iErr, szErrBuf, 1);
(234)
(235)       /* connection start */
(236)       ConnectionStart(pConnection, &iErr, szErrBuf);
(237)       check_error(iErr, szErrBuf, 1);
(238)
(239)       /* create xa id */
(240)       pXid = XACreateXid((char*)MSCLIENT_DLL_NAME, &iErr, szErrBuf);
(241)       check_error(iErr, szErrBuf, 1);
(242)
(243)       /* associate the global transaction with the resource */
(244)       XAResourceStart(pXATopicResource, pXid, SBYN_TMNOFLAGS, &iErr,
(245)       szErrBuf);
(246)       check_error(iErr, szErrBuf, 1);
(247)
(248)       /* create a meesage */
(249)       pMessage = SessionCreateTextMessage(pXATopicSession, &iErr,
(250)       szErrBuf);
(251)       check_error(iErr, szErrBuf, 1);
(252)
(253)       /* set mode to r/w */
(254)       ClearBody(pMessage, &iErr, szErrBuf);
(255)       check_error(iErr, szErrBuf, 1);
(256)
(257)       /* write bytes */
(258)       SetText(pMessage, (char*)szText, &iErr, szErrBuf);
(259)       check_error(iErr, szErrBuf, 1);
(260)
(261)       /* publish message */
(262)       printf("Sending %d messages\n", iNumMessages);
(263)       PublishMessage(pTopicPublisher, pMessage, iNumMessages);
(264)
(265)       /* xaEnd */
(266)       XAResourceEnd(pXATopicResource, pXid, SBYN_TMSUCCESS, &iErr,
(267)       szErrBuf);
(268)       check_error(iErr, szErrBuf, 1);
(269)
(270)       /* ================================================
(271)        * Prepare-Rollback
(272)        * ================================================
(273)        */
(274)       /* xaPrepare */
(275)       XAResourcePrepare(pXATopicResource, pXid, &iErr, szErrBuf);
(276)       check_error(iErr, szErrBuf, 1);
(277)
(278)       /* xaRollBack */
(279)       printf("Rolling back %d message\n", iNumMessages);
(280)       XAResourceRollback(pXATopicResource, pXid, &iErr, szErrBuf);
(281)       check_error(iErr, szErrBuf, 1);
(282)
(283)
(284)       /* ================================================
(285)        * Prepare-Commit
(286)        * ================================================
(287)        */
```

```
(288)
(289)      /* xa start */
(290)      XAResourceStart(pXATopicResource, pXid, SBYN_TMNOFLAGS, &iErr,
(291)      szErrBuf);
(292)      check_error(iErr, szErrBuf, 1);
(293)
(294)      /* send message */
(295)      printf("Sending %d messages\n", iNumMessages);
(296)      PublishMessage(pTopicPublisher, pMessage, iNumMessages);
(297)
(298)      /* xaEnd */
(299)      XAResourceEnd(pXATopicResource, pXid, SBYN_TMSUCCESS, &iErr,
(300)      szErrBuf);
(301)      check_error(iErr, szErrBuf, 1);
(302)
(303)      /* xaPrepare */
(304)      if (SBYN_XA_OK != XAResourcePrepare(pXATopicResource, pXid, &iErr,
(305)      szErrBuf))
(306)      {
(307)           printf("ERROR: XAResourcePrepare failed\n");
(308)      }
(309)      check_error(iErr, szErrBuf, 1);
(310)
(311)      /* xa commit */
(312)      printf("Session Commit...\n");
(313)      XAResourceCommit(pXATopicResource, pXid, TRUE, &iErr, szErrBuf);
(314)      check_error(iErr, szErrBuf, 1);
(315)
(316)      /* Close and clean up. */
(317)      TopicPublisherClose(pTopicPublisher, &iErr, szErrBuf);
(318)      check_error(iErr, szErrBuf, 1);
(319)      SessionClose(pXATopicSession, &iErr, szErrBuf);
(320)      check_error(iErr, szErrBuf, 1);
(321)      ConnectionClose(pConnection, &iErr, szErrBuf);
(322)      check_error(iErr, szErrBuf, 1);
(323)      DeleteMessage(pMessage, &iErr, szErrBuf);
(324)      DeleteTopicPublisher(pTopicPublisher, &iErr, szErrBuf);
(325)  DeleteXAResource(pXATopicResource, &iErr, szErrBuf);
(326)      DeleteXid(pXid, &iErr, szErrBuf);
(327)      DeleteSession(pXATopicSession, &iErr, szErrBuf); /* delete session
(328)      & resource */
(329)      DeleteDestination(pTopic, &iErr, szErrBuf);
(330)      DeleteConnection(pConnection, &iErr, szErrBuf);
(331)      DeleteXATopicConnectionFactory(pXATcf, &iErr, szErrBuf);
(332) }
(333)
(334) /*
(335)  * ========================================================
(336)  * Topic Subscriber
(337)  * This routine demonstrates how to subsribe a message from a
(338)  *  topic.
(339)  * ========================================================
(340) */
(341) void XATopicSub()
(342) {
(343)      SBYN_XATopicConnectionFactory* pXATcf = 0;
(344)      SBYN_Connection*               pConnection = 0;
(345)      SBYN_Session*                  pXATopicSession = 0;
(346)      SBYN_Session*                  pTopicSession = 0;
(347)      SBYN_XAResource*               pTopicResource = 0;
(348)      SBYN_Destination*              pTopic = 0;
(349)      SBYN_Message*                  pReceivedMessage = 0;
(350)      SBYN_XAResource*               pXATopicResource = 0;
(351)      SBYN_TopicSubscriber*          pTopicSubscriber = 0;
```

```
(352)      SBYN_Message*                  pMessage = 0;
(353)      SBYN_Xid*                      pXid = 0;
(354)     char                           pTopicName[] = "eGateXAPubSubSample";
(355)      int                             iNumReceived = 0;
(356)      char                            szUserInput[8];
(357)      /* create XA connection factory */
(358)     pXATcf = CreateXATopicConnectionFactory(spHostName, susPort, &iErr,
(359)      szErrBuf);
(360)      check_error(iErr, szErrBuf, 1);
(361)
(362)      /* create XA connection */
(363)      pConnection = CreateXATopicConnection(pXATcf, &iErr, szErrBuf);
(364)      check_error(iErr, szErrBuf, 1);
(365)
(366)      /* set client ID */
(367)      ConnectionSetClientID(pConnection, (char*)"eGate{7E527692-770A-
(368)      11D5-B139-3456789}", &iErr, szErrBuf);
(369)      check_error(iErr, szErrBuf, 1);
(370)
(371)      /* create XA session */
(372)      pXATopicSession = XAConnectionCreateXATopicSession(pConnection,
(373)      &iErr, szErrBuf);
(374)      check_error(iErr, szErrBuf, 1);
(375)
(376)      /* get session */
(377)      pTopicSession = XASessionGetTopicSession(pXATopicSession, &iErr,
(378)      szErrBuf);
(379)      check_error(iErr, szErrBuf, 1);
(380)
(381)      /* get XA resource */
(382)      pXATopicResource = XASessionGetXAResource(pXATopicSession, &iErr,
(383)      szErrBuf);
(384)      check_error(iErr, szErrBuf, 1);
(385)
(386)      /* create a Topic */
(387)      pTopic = SessionCreateTopic(pTopicSession, pTopicName, &iErr,
(388)      szErrBuf);
(389)      check_error(iErr, szErrBuf, 1);
(390)
(391)      /* create a subscriber */
(392)      //pTopicSubscriber = SessionCreateDurableSubscriber(pTopicSession,
(393)      pTopic, (char*)"XATopicSubscriber", &iErr, szErrBuf);
(394)      pTopicSubscriber = SessionCreateSubscriber(pTopicSession, pTopic,
(395)      &iErr, szErrBuf);
(396)      check_error(iErr, szErrBuf, 1);
(397)
(398)      /* connection start */
(399)      ConnectionStart(pConnection, &iErr, szErrBuf);
(400)      check_error(iErr, szErrBuf, 1);
(401)
(402)      /* start xa resource */
(403)      pXid = XACreateXid((char*)MSCLIENT_DLL_NAME, &iErr, szErrBuf);
(404)      check_error(iErr, szErrBuf, 1);
(405)
(406)
(407)
(408)      /* Receive all the messages on the topic and return the number of
(409)      messages received */
(410)      //iNumReceived = SubscriberReceiveMessage(pTopicSubscriber);
(411)      printf("Receiving messages...\n");
(412)      do {
(413)   int iMsgCount = 0;
(414)           SBYN_Message* pRMsg = 0;
(415)           /* associate the global transaction with the resource */
```

```
(416)   XAResourceStart(pXATopicResource, pXid, SBYN_TMNOFLAGS, &iErr,
(417)   szErrBuf);
(418)   check_error(iErr, szErrBuf, 1);
(419)           while(iMsgCount < iNumMessages){
(420)                pRMsg = TopicSubscriberReceive(pTopicSubscriber, &iErr,
(421)                szErrBuf);
(422)                printf("Received Message %d\n", iMsgCount);
(423)                iMsgCount++;
(424)           }
(425)
(426)           /* xaEnd */
(427)   XAResourceEnd(pXATopicResource, pXid, SBYN_TMSUCCESS, &iErr,
(428)   szErrBuf);
(429)   check_error(iErr, szErrBuf, 1);
(430)           XAResourceCommit(pXATopicResource, pXid, TRUE, &iErr,
(431)   szErrBuf);
(432)           printf("Enter 'r' for receiving more message, 'q' for
(433)   exit\n");
(434)           scanf("%s", szUserInput);
(435)           iMsgCount = 0;
(436)      } while(szUserInput[0] != 'q' && szUserInput[0] != 'Q');
(437)
(438)      /* Session commit */
(439)      SessionCommit(pTopicSession,  &iErr, szErrBuf);
(440)
(441)      /* close and delete objects */
(442)      TopicSubscriberClose(pTopicSubscriber, &iErr, szErrBuf);
(443)      SessionUnsubscribe(pXATopicSession, (char*)"TopicSubscriber",
(444)      &iErr, szErrBuf);
(445)      SessionClose(pXATopicSession, &iErr, szErrBuf);
(446)      ConnectionClose(pConnection, &iErr, szErrBuf);
(447)
(448)      /* delete objects */
(449)      DeleteDestination(pTopic, &iErr, szErrBuf);
(450)      DeleteTopicSubscriber(pTopicSubscriber, &iErr, szErrBuf);
(451)  DeleteXAResource(pXATopicResource, &iErr, szErrBuf);
(452)      DeleteXid(pXid, &iErr, szErrBuf);
(453)      DeleteSession(pXATopicSession, &iErr, szErrBuf); /* delete XA
(454)      resource and session */
(455)      DeleteConnection(pConnection, &iErr, szErrBuf);
(456)      DeleteXATopicConnectionFactory(pXATcf, &iErr, szErrBuf);

    }
```

Appendix B

C++ Sample Code

The C++ sample code is accessed from the sample files downloaded from the Documentation tab of the installer (see **"About the C and C++ Samples" on page 245** for more information). The samples were built using the compiler shown for the Windows operating system in **Table 2 on page 19**. This appendix lists the sample code provided for the types of messaging listed below.

What's in This Appendix

B.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 (JMSException &e)
(174)     {
(175)        printf("JMS error: %s\n", e.getMessage());
(176)     }
(177) }
(178)
(179) /*
(180)  * ============================================================
(181)  * Topic Subscriber
(182)  * This routine demonstrates how to subsribe a message from
(183)  * a topic.
(184)  * ============================================================
(185) */
(186) static void subscriber() {
(187)     char                    szUserInput[80];
(188)     char                    pTopicName[] = "eGatePubSubSample";
(189)
(190)     try {
(191)     /* create a topic connection factory*/
(192)     TopicConnectionFactory*     pTopicConnectionFactory =
(193)         LookupTopicConnectionFactory(MSCLIENT_DLL_NAME, "", spHostName,
(194)         susPort, 0, 0);
(195)
(196)     /* create a topic connection */
(197)     TopicConnection* pTopicConnection = pTopicConnectionFactory-
(198)     >createTopicConnection();
(199)
(200)     /* set client ID */
(201)     pTopicConnection->setClientID("TopicSubscriber");
(202)
(203)     /* start connection */
(204)     pTopicConnection->start();
(205)
(206)     /* create a topic session */
(207)     TopicSession* pTopicSession = pTopicConnection-
(208)     >createTopicSession(true, Session::AUTO_ACKNOWLEDGE);
(209)
(210)
(211)     /* create a topic */
(212)     WString wsTopicName(pTopicName);
(213)     Topic* pTopic = pTopicSession->createTopic(wsTopicName);
(214)
(215)     /* create a subscriber */
(216)     TopicSubscriber* pTopicSubscriber = pTopicSession-
(217)     >createDurableSubscriber(pTopic, (char*)"TopicPublisher");
(218)
```

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

B.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 (JMSException &e)
(174)      {
(175)          printf("JMS error: %s\n", e.getMessage());
(176)      }
(177) }
(178)
(179) /*
(180)  * ============================================================
(181)  *  Queue Receiver
(182)  *  This routine demonstrates how to receive a message from a
(183)  *  queue.
(184)  * ============================================================
(185) */
(186) static void receiver() {
(187)      char                    szUserInput[80];
(188)      char                    pQueueName[] = "eGateP2PSample";
(189)
(190)      try {
(191)      /* Create a queue connection factory */
(192)      QueueConnectionFactory* pQueueConnectionFactory =
(193)           LookupQueueConnectionFactory(MSCLIENT_DLL_NAME, "",
      spHostName, susPort, 0, 0);
(194)
(195)      /* Create a queue connection */
(196)      QueueConnection* pQueueConnection = pQueueConnectionFactory-
      >createQueueConnection();
(197)
(198)      /* Set the client ID */
(199)      pQueueConnection->setClientID("RECEIVER");
(200)
(201)      /* Start the connection */
(202)      pQueueConnection->start();
(203)
(204)      /* Create a queue session */
(205)      QueueSession* pQueueSession = pQueueConnection-
      >createQueueSession(true, Session::CLIENT_ACKNOWLEDGE);
(206)
(207)      /* Create a queue */
(208)      WString wsQueueName(pQueueName);
(209)      Queue* pQueue = pQueueSession->createQueue(wsQueueName);
(210)
```

```
(211)      /* Create a queue receiver */
(212)      QueueReceiver* pQueueReceiver = pQueueSession-
      >createReceiver(pQueue);
(213)
(214)      printf("Waiting for message ... \n");
(215)      TextMessage* pReceivedTextMessage;
(216)      do {
(217)           /* waiting for incoming messages */
(218)           Message* pReceivedMessage = pQueueReceiver->receive();
(219)           pReceivedTextMessage = DYNAMIC_CAST(TextMessage,
      pReceivedMessage);
(220)           if (pReceivedTextMessage){
(221)                WString wsJMSType = pReceivedTextMessage->getJMSType();
(222)                WString wsText = pReceivedTextMessage->getText();
(223)                string  strText;
(224)                strText = MsLocale::WideStringToString(wsText).c_str();
(225)                cout << "Received Text Message " << strText << endl;
(226)           }
(227)           printf("Enter 'r' for receiving more message, 'q' for
      exit\n");
(228)           scanf("%s", szUserInput);
(229)      } while ( szUserInput[0] != 'q' && szUserInput[0] != 'Q' );
(230)
(231)      /* close and delete objects */
(232)      pQueueReceiver->close();
(233)      pQueueSession->close();
(234)      pQueueConnection->close();
(235)      delete(pReceivedTextMessage);
(236)      delete(pQueueReceiver);
(237)      delete(pQueueSession);
(238)      delete(pQueueConnection);
(239)      delete(pQueueConnectionFactory);
(240)      }
(241)      catch (JMSException &e)
(242)      {
(243)          printf("JMS error: %s\n", e.getMessage());
(244)      }
(245) }
```

B.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)
```

B.4 Message Selectors Using C++

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

```
(31)  #if defined(OS400)
(32)  extern char *optarg;
(33)  #endif
(34)
(35)  #if defined(__gnu__)
(36)  #include <getopt.h>
(37)  #endif
(38)
(39)  #define                 MSCLIENT_DLL_NAME     "stc_msclient.dll"
(40)  char                    optionProducer[] = "[ -u ] [ -p port ]
(41)                          [ -h hostname ]";
(42)  char                    optionConsumer[] = "[ -c ] [ -p port ]
(43)                          [ -h hostname ]";
(44)  char                    optdescription[] = "\t-u run as a
(45)                          producer\n\t-c run as a consumer\n\t-p
(46)                          port number\n\t-h hostname\n";
(47)  static char             localHost[] = "localhost";
(48)  static unsigned short   susPort = 24053; /* default port number */
(49)  unsigned long           sulMessageSize = 16; /* default host name */
(50)  static char*            spHostName;
(51)  static char PROP_NAME[] = "property";
(52)
(53)
(54)  static void selector_publisher();
(55)  static void selector_subscriber();
(56)
(57)
(58)  int main(int argc, char *argv[]) {
(59)       int           c;
(60)       char          cOption = 0;
(61)
(62)       spHostName = localHost;
(63)
(64)       while((c = getopt(argc, argv, ":p:h:P:H:cCUu")) != -1) {
(65)            switch(c){
(66)            case 'p':
(67)            case 'P':
(68)                 susPort = atoi(optarg);  /* setup the port number */
(69)                 break;
(70)            case 'h':
(71)            case 'H':
(72)                 spHostName = optarg;     /* setup the hostname */
(73)                 break;
(74)            case 'U':
(75)            case 'u':
(76)                 cOption = 'u';           /* run as a producer */
(77)                 break;
(78)            case 'c':
(79)            case 'C':
(80)                 cOption = 'c';           /* run as a consumer */
(81)                 break;
(82)            case ':':
(83)            case '?':
(84)                 printf("\nSYNOPSIS\n");
(85)                 printf("%s %s\n", argv[0], optionProducer);
(86)                 printf("%s %s\n", argv[0], optionConsumer);
(87)                 printf("%s\n", optdescription);
(88)                 exit(1);
(89)                 break;
(90)            }
(91)       }
(92)
(93)       if (cOption == 'u'){
(94)            selector_publisher();          /* invoke producer */
```

```
(95)       } else if (cOption == 'c'){
(96)            selector_subscriber();         /* invoke consumer */
(97)       } else {
(98)            printf("\nSYNOPSIS\n");
(99)            printf("%s %s\n", argv[0], optionProducer);
(100)           printf("%s %s\n", argv[0], optionConsumer);
(101)           printf("%s\n", optdescription);
(102)           exit(1);
(103)      }
(104)  return 0;
(105) }
(106)
(107) /*
(108)  * ========================================================
(109)  * Topic Publisher
(110)  * This routine demonstrates how to publish to a topic.
(111)  * ========================================================
(112) */
(113) static void selector_publisher(){
(114)      int                                 ii;
(115)      static char                         pTopicName[] = "Selector";
(116)
(117)      try {
(118)      /* Create a topic factory. */
(119)      TopicConnectionFactory* pTopicConnectionFactory =
(120)           LookupTopicConnectionFactory(MSCLIENT_DLL_NAME, "",
(121)           spHostName, susPort, 0, 0);
(122)
(123)      /* Create a topic connection. */
(124)      TopicConnection* pTopicConnection = pTopicConnectionFactory-
(125)      >createTopicConnection();
(126)
(127)      /* Set the client ID. */
(128)      pTopicConnection->setClientID("TopicPublisher");
(129)
(130)      /* Start the connection. */
(131)      pTopicConnection->start();
(132)
(133)      /* Create a topic session. */
(134)      TopicSession* pTopicSession = pTopicConnection-
(135)      >createTopicSession(true, Session::AUTO_ACKNOWLEDGE);
(136)
(137)      /* Create a topic. */
(138)      WString wsTopicName(pTopicName);
(139)      Topic* pTopic = pTopicSession->createTopic(wsTopicName);
(140)
(141)      /* Create a topic publisher. */
(142)      TopicPublisher* pTopicPublisher = pTopicSession-
(143)      >createPublisher(pTopic);
(144)
(145)
(146)      /* Set delivery mode as persistent */
(147)      pTopicPublisher->setDeliveryMode(DeliveryMode::PERSISTENT);
(148)
(149)
(150)      /* publish 10 messages to the topic */
(151)      TextMessage* msglist[10];
(152)      for(ii=0; ii<10 ;ii++){
(153)           int index;
(154)           char buf[80];
(155)
(156)           /* Create a text message. */
(157)           msglist[ii] = pTopicSession->createTextMessage();
(158)
```

```
(159)           /* Clear the body (payload) of the message. */
(160)           msglist[ii]->clearBody();
(161)           index = ii % 10;
(162)           sprintf(buf, "%d", index);
(163)
(164)           /* Set the string property */
(165)           msglist[ii]->setStringProperty(PROP_NAME, buf);
(166)
(167)           /* Copy in the text to be sent. */
(168)           msglist[ii]->setText("This is a text message");
(169)
(170)           /* Publish the message. */
(171)           pTopicPublisher->publish(msglist[ii]);
(172)           printf("... Published 1 message with property %s = %d\n",
(173)           PROP_NAME, ii);
(174)      }
(175)
(176)      /* Commit the session. */
(177)      pTopicSession->commit();
(178)
(179)
(180)      /* close and delete objects */
(181)      pTopicPublisher->close();
(182)      pTopicSession->close();
(183)      pTopicConnection->close();
(184)      for (ii = 0; ii < 10; ii++){
(185)           delete(msglist[ii]);
(186)      }
(187)      delete(pTopicPublisher);
(188)      delete(pTopicSession);
(189)      delete(pTopicConnection);
(190)      delete(pTopicConnectionFactory);
(191)      }
(192)      catch (JMSException &e)
(193)      {
(194)          printf("JMS error: %s\n", e.getMessage());
(195)      }
(196) }
(197)
(198) /*
(199)  * ============================================================
(200)  * Topic Subscriber
(201)  * This routine demonstrates how to subsribe a message from a
(202)  * topic.
(203)  * ============================================================
(204)  */
(205) static void selector_subscriber(){
(206)      char                 selectorString[80];
(207)      char                 selectorSubscriberName[80];
(208)      int                  selector = 7;
(209)      char*                selectorName;
(210)      char                 pTopicName[] = "eGateSelector";
(211)
(212)      try {
(213)      /* create a topic connection factory */
(214)      TopicConnectionFactory* pTopicConnectionFactory =
(215)           LookupTopicConnectionFactory(MSCLIENT_DLL_NAME, "",
(216)      spHostName, susPort, 0, 0);
(217)
(218)      /* Create a topic connection. */
(219)      TopicConnection* pTopicConnection = pTopicConnectionFactory-
(220)      >createTopicConnection();
(221)
(222)      /* Set the client ID. */
```

```
(223)      pTopicConnection->setClientID("Publisher");
(224)
(225)      /* Start the connection. */
(226)      pTopicConnection->start();
(227)
(228)      /* Create a topic session. */
(229)      TopicSession* pTopicSession = pTopicConnection-
      >createTopicSession(true, Session::AUTO_ACKNOWLEDGE);
(230)
(231)      /* Create a topic. */
(232)      WString wsTopicName(pTopicName);
(233)      Topic* pTopic = pTopicSession->createTopic(wsTopicName);
(234)
(235)
(236)      /* Create subscriber  with selector*/
(237)      sprintf(selectorString, "%s = '%d'", PROP_NAME, selector);
(238)      sprintf(selectorSubscriberName, "SelectorSubscriber%d", selector);
(239)      TopicSubscriber* pTopicSubscriber = pTopicSession-
(240)      >createDurableSubscriber(pTopic, selectorSubscriberName,
(241)      selectorString, 0);
(242)
(243)      /* Get message using selector */
(244)      selectorName = pTopicSubscriber->getMessageSelector();
(245)      printf("using selector: %s\n", selectorName);
(246)
(247)      Message* pMessage;
(248)      for (pMessage = pTopicSubscriber->receive();
(249)           pMessage != 0;
(250)           pMessage = pTopicSubscriber->receive(1000))
(251)      {
(252)           string strProperty =  MsLocale::WideStringToString(pMessage-
(253)           >getStringProperty(PROP_NAME)).c_str();
(254)           cout << "Received 1 message with " << PROP_NAME << " = " <<
(255)           strProperty << endl;;
(256)           delete(pMessage);
(257)      }
(258)
(259)      /* Session commit */
(260)      pTopicSession->commit();
(261)
(262)      /* Close and delete objects */
(263)      pTopicSubscriber->close();
(264)      pTopicSession->close();
(265)      pTopicConnection->close();
(266)      delete(pTopicSubscriber);
(267)      delete(pTopicSession);
(268)      delete(pTopicConnection);
(269)      delete(pTopicConnectionFactory);
(270)      }
(271)      catch (JMSException &e)
(272)      {
(273)          printf("JMS error: %s\n", e.getMessage());
(274)      }
(275) }
```

B.5 XA Publish/Subscribe Messaging Using C++

```
(1)    *------------------------------------------------------------------
(2)    * Sample code to demonstrate JMS Pub/Sub in XA.
(3)    *------------------------------------------------------------------
(4)    *
```

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

```
(65)                 break;
(66)            case 'h':
(67)            case 'H':
(68)                 spHostName = optarg;     /* setup the hostname */
(69)                 break;
(70)            case 'U':
(71)            case 'u':
(72)                 cOption = 'u';           /* run as a producer */
(73)                 break;
(74)            case 'c':
(75)            case 'C':
(76)                 cOption = 'c';           /* run as a consumer */
(77)                 break;
(78)            case ':':
(79)            case '?':
(80)                 printf("\nSYNOPSIS\n");
(81)                 printf("%s %s\n", argv[0], optionProducer);
(82)                 printf("%s %s\n", argv[0], optionConsumer);
(83)                 printf("%s\n", optdescription);
(84)                 exit(1);
(85)                 break;
(86)            }
(87)       }
(88)
(89)       if (cOption == 'u'){
(90)            XATopicPub();          /* invoke producer */
(91)       } else if (cOption == 'c'){
(92)            XATopicSub();         /* invoke consumer */
(93)       } else {
(94)            printf("\nSYNOPSIS\n");
(95)            printf("%s %s\n", argv[0], optionProducer);
(96)            printf("%s %s\n", argv[0], optionConsumer);
(97)            printf("%s\n", optdescription);
(98)            exit(1);
(99)       }
(100)      return 0;
(101) }
(102)
(103) /*
(104)  * ========================================================
(105)  * Publish Message
(106)  *   This routine publishes iNumMessages to the topic
(107)  * ========================================================
(108) */
(109) static void PublishMessage(TopicPublisher* pPublisher, Message*
      pMessage, int iNumMessages)
(110) {
(111)      int ii;
(112)      for ( ii = 0; ii < iNumMessages; ii++){
(113)           pMessage->setIntProperty("Sequence", ii);
(114)           printf("Sending Message: Sequence number %d\n", ii);
(115)           pPublisher->publish(pMessage);
(116)      }
(117) }
(118)
(119)
(120) /*
(121)  * ========================================================
(122)  * Receive Message
(123)  *   This routine block on receiving message for maximum iWait
(124)  *   seconds before return.
(125)  * ========================================================
(126) */
```

```
(127) static int SubscriberReceiveMessage(TopicSubscriber* pSub,
      TopicSession* pSession)
(128) {
(129)      int iMsgCount = 0;
(130)      Message* pRMsg = 0;
(131)      char szUserInput[8];
(132)      printf("Waiting for message ... \n");
(133)      do {
(134)           pRMsg = pSub->receive();
(135)           printf("Received Message %d\n", iMsgCount);
(136)           iMsgCount++;
(137)           if (iMsgCount >= iNumMessages){
(138)                printf("Enter 'r' for receiving more message, 'q' for
      exit\n");
(139)                scanf("%s", szUserInput);
(140)                pSession->commit();
(141)                iMsgCount = 0;
(142)           }
(143)
(144)      } while(szUserInput[0] != 'q' && szUserInput[0] != 'Q');
(145)      return iMsgCount;
(146) }
(147)
(148) /*
(149)  * ========================================================
(150)  * Topic Publisher
(151)  *   This routine demonstrates how to publish to a topic.
(152)  * ========================================================
(153) */
(154) void XATopicPub()
(155) {
(156)      char                          pTopicName[] = "XAPubSubSample";
(157)      Xid *pXid;
(158)
(159)      try {
(160)      /* Create a topic factory. */
(161)      XATopicConnectionFactory* pXATopicConnectionFactory =
(162)           LookupXATopicConnectionFactory(MSCLIENT_DLL_NAME, "",
(163)      spHostName, susPort, 0, 0);
(164)
(165)      /* Create a topic connection. */
(166)      XATopicConnection* pXATopicConnection = pXATopicConnectionFactory-
(167)      >createXATopicConnection();
(168)
(169)      /* set client ID */
(170)      pXATopicConnection->setClientID("eGate{7E527692-770A-11D5-B139-
(171)      935EB6E85DBD}");
(172)
(173)      /* create XA session */
(174)      XATopicSession* pXATopicSession = pXATopicConnection-
(175)      >createXATopicSession();
(176)
(177)      /* get session */
(178)      TopicSession* pTopicSession = pXATopicSession->getTopicSession();
(179)
(180)      /* get XA resource */
(181)      XAResource* pXATopicResource = pXATopicSession->getXAResource();
(182)
(183)      /* create a Topic */
(184)      Topic* pTopic = pTopicSession->createTopic(pTopicName);
(185)
(186)      /* create a publisher */
(187)      TopicPublisher* pTopicPublisher = pTopicSession-
(188)      >createPublisher(pTopic);
```

```
(189)
(190)       /* connection start */
(191)       pXATopicConnection->start();
(192)
(193)       /* create xa id */
(194)       pXid = CreateXid(MSCLIENT_DLL_NAME);
(195)
(196)       /* associate the global transaction with the resource */
(197)       pXATopicResource->start(pXid, XAResource::TMNOFLAGS);
(198)
(199)       /* create a meesage */
(200)       TextMessage* pMessage = pXATopicSession->createTextMessage();
(201)
(202)       /* set mode to r/w */
(203)       pMessage->clearBody();
(204)
(205)       /* write bytes */
(206)       pMessage->setText((char*)szText);
(207)
(208)       /* publish message */
(209)       printf("Sending %d messages\n", iNumMessages);
(210)       pTopicPublisher->publish(pMessage);
(211)
(212)       /* xaEnd */
(213)       pXATopicResource->end(pXid, XAResource::TMSUCCESS);
(214)
(215)       /* ====================================================
(216)        * Prepare-Rollback
(217)        * ====================================================
(218)        */
(219)       /* xaPrepare */
(220)       pXATopicResource->prepare(pXid);
(221)
(222)       /* xaRollBack */
(223)       printf("Rolling back %d message\n", iNumMessages);
(224)       pXATopicResource->rollback(pXid);
(225)
(226)
(227)       /* ====================================================
(228)        * Prepare-Commit
(229)        * ====================================================
(230)        */
(231)
(232)       /* xa start */
(233)       pXATopicResource->start(pXid, XAResource::TMNOFLAGS);
(234)
(235)       /* send message */
(236)       printf("Sending %d messages\n", iNumMessages);
(237)       PublishMessage(pTopicPublisher, pMessage, iNumMessages);
(238)
(239)       /* xaEnd */
(240)       pXATopicResource->end(pXid, XAResource::TMSUCCESS);
(241)
(242)       /* xaPrepare */
(243)       if (XAResource::XA_OK != pXATopicResource->prepare(pXid))
(244)       {
(245)            printf("ERROR: XAResourcePrepare failed\n");
(246)       }
(247)
(248)       /* xa commit */
(249)       printf("Resource Commit...\n");
(250)       pXATopicResource->commit(pXid, true);
(251)
(252)       /* Close and clean up. */
```

```
(253)      pTopicPublisher->close();
(254)      pXATopicSession->close();
(255)      pXATopicConnection->close();
(256)      delete(pMessage);
(257)      delete(pTopicPublisher);
(258)      delete(pXid);
(259)      delete(pXATopicSession);
(260)      delete(pTopic);
(261)      delete(pXATopicConnection);
(262)      delete(pXATopicConnectionFactory);
(263)      }
(264)      catch (JMSException &e)
(265)      {
(266)          printf("JMS error: %s\n", e.getMessage());
(267)      }
(268) }
(269)
(270) /*
(271)  * ============================================================
(272)  *      Topic Subscriber
(273)  *  This routine demonstrates how to subsribe a message from a
(274)  *  topic.
(275)  * ============================================================
(276) */
(277) void XATopicSub()
(278) {
(279)      char                          pTopicName[] = "eGateXAPubSubSample";
(280)      int                             iNumReceived = 0;
(281)      char                            szUserInput[8];
(282)
(283)      try {
(284)      /* Create a topic factory. */
(285)      XATopicConnectionFactory* pXATopicConnectionFactory =
(286)           LookupXATopicConnectionFactory(MSCLIENT_DLL_NAME, "",
(287)      spHostName, susPort, 0, 0);
(288)
(289)      /* Create a topic connection. */
(290)      XATopicConnection* pXATopicConnection = pXATopicConnectionFactory-
(291)      >createXATopicConnection();
(292)
(293)
(294)      /* set client ID */
(295)      pXATopicConnection->setClientID("eGate{7E527692-770A-11D5-B139-
(296)      3456789}");
(297)
(298)      /* create XA session */
(299)      XATopicSession* pXATopicSession = pXATopicConnection-
(300)      >createXATopicSession();
(301)
(302)      /* get session */
(303)      TopicSession* pTopicSession = pXATopicSession->getTopicSession();
(304)
(305)      /* get XA resource */
(306)      XAResource* pXATopicResource = pXATopicSession->getXAResource();
(307)
(308)      /* create a Topic */
(309)      Topic* pTopic = pTopicSession->createTopic(pTopicName);
(310)
(311)      /* create a subscriber */
(312)      TopicSubscriber* pTopicSubscriber = pTopicSession-
(313)      >createDurableSubscriber(pTopic, (char*)"XATopicSubscriber");
(314)
(315)      /* connection start */
(316)      pXATopicConnection->start();
```

```
(317)
(318)       /* start xa resource */
(319)       Xid* pXid = CreateXid(MSCLIENT_DLL_NAME);
(320)
(321)
(322)       int iMsgCount = 0;
(323)       do {
(324)            Message* pRMsg = 0;
(325)            /* associate the global transaction with the resource */
(326)            pXATopicResource->start(pXid, XAResource::TMNOFLAGS);
(327)            while(iMsgCount < iNumMessages){
(328)                 pRMsg = pTopicSubscriber->receive();
(329)                 printf("Received Message %d\n", iMsgCount);
(330)                 iMsgCount++;
(331)            }
(332)            /* xaEnd */
(333)            pXATopicResource->end(pXid, XAResource::TMSUCCESS);
(334)            pXATopicResource->commit(pXid, true);
(335)            printf("Enter 'r' for receiving more message, 'q' for
(336)            exit\n");
(337)            scanf("%s", szUserInput);
(338)
(339)            //pTopicSession->commit();
(340)            iMsgCount = 0;
(341)       } while(szUserInput[0] != 'q' && szUserInput[0] != 'Q');
(342)
(343)
(344)
(345)       /* close and delete objects */
(346)       pTopicSubscriber->close();
(347)       //pTopicSession->unsubscribe((char*)"TopicSubscriber");
(348)       pXATopicSession->close();
(349)       pXATopicConnection->close();
(350)
(351)       /* delete objects */
(352)       delete(pTopic);
(353)       delete(pTopicSubscriber);
(354)       delete(pXid);
(355)       delete(pXATopicSession);
(356)       delete(pXATopicConnection);
(357)       delete(pXATopicConnectionFactory);
(358)       }
(359)       catch (JMSException &e)
(360)       {
(361)           printf("JMS error: %s\n", e.getMessage());
(362)       }
       }
```

Index

Symbols

A

B

C

D

E

F

G

H

I

J

L

M

N

O

P

Q

R

S

T

U

W

X