SUN SEEBEYOND

eGATE™ API KIT FOR JMS IQ MANAGER (COBOL EDITION)

Release 5.1.3

Sun microsystems

Part Number: 820-0934-10

Version 20070412115248

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 COBOL applications that connect to Sun Java™ Composite Platform Suite (CAPS) Projects via Java™ Message Service (JMS).

1.1.1 What's in This Document

This document includes the following chapters:

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

1.1.2 Intended Audience

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

1.1.3 Text Conventions

The following conventions are observed throughout this document.

Table 1 Text Conventions

Text Convention	Used For	Examples
Bold	Names of buttons, files, icons, parameters, variables, methods, menus, and objects	▪ Click **OK**. ▪ On the **File** menu, click **Exit**. ▪ Select the **eGate.sar** file.
`Monospaced`	Command line arguments, code samples; variables are shown in ***bold italic***	`java -jar ***filename***.jar`
Blue bold	Hypertext links within document	See **Related Documents** on page 6
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 and related products, 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 Administration 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

The eGate API Kit for COBOL can be installed on IBM z/OS V1.3. The initial installation is performed using the Sun Java Composite Application Suite Installer (hereafter, Suite Installer), which runs on Windows operating systems. Unlike the eGate API Kits for Windows or Unix operating systems, the API kit for COBOL is not dependent on the operating system of the machine that hosts the Java CAPS Repository.

2.2 System Requirements

The eGate API Kit for COBOL requires

- An MVS system with free space for the two datasets
- FTP access
- A COBOL compiler
- A user login ID for the MVS system
- CICS TS 3.1 or later, if you are using the eGate API Kit for COBOL with CICS

Important: The library files are very large by MVS standards. You will most likely need to increase your region size for CICS.

2.3 Installing the eGate API Kit

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

2.3.1 Before You Begin

This version of the eGate API Kit for COBOL is compatible with both version 5.0.5 and 5.1.3 of Java CAPS. Before you install the eGate API Kit, install and download the following items using the Suite Installer:

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

2.3.2 Installing the eGate API Kit to the Repository

To access the installation files for the eGate API Kit, you need to upload the files to an eGate Repository. This makes the kit available to developers through the Java CAPS Repository. The procedures below describe how to install the following items for the eGate API Kit:

- the installation files
- the Project samples
- the documentation

To install the 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 API Kit COBOL** (to install the eGate API Kit software)
   - **Documentation > eGateAPIKitDocs** (optional—to install the eGate API Kit documentation and samples)
4. In the **Administrator > Upload** page, select the following items as prompted and click **Next** after each SAR file is selected:
   - **eGate_APIKit_COBOL_ZOS.sar**
   - **eGateAPIKitDocs.sar**

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

5 To view the documentation:

A Click the **Documentation** tab, and then click **Add-ons**.

B In the product list, click **Sun SeeBeyond eGate(TM) API Kit**.

C Click the icon next to the PDF document or HTML help system you want to view.

2.3.3 Downloading the eGate API Kit to a Local Machine

Once the eGate API Kit is installed to the Java CAPS Repository, the kit is available to be downloaded by developers to their local machines so they can work with the libraries and samples.

To download the eGate API Kit

1 On the **Downloads** page of the Suite Installer, select **API kit for COBOL ZOS**, select a location for the **.zip** file to be saved, and then extract the file.

2 To download the sample Projects:

A Click the **Documentation** tab, and then click **Add-ons**.

B In the product list, click **Sun SeeBeyond eGate(TM) API Kit**.

C Click the zip file icon next to **Sample Projects**, and then click **Open**.

D Extract the files to a temporary directory on your computer.

E Navigate to the temporary directory.

F Extract **ProjectSamples.zip**.

For information about using the samples, refer to **Chapter 5 "Working with the COBOL API Samples"**.

2.3.4 Installing to the MVS System

After you have the eGate API Kit installation files on your computer, perform the following procedure install the files on the MVS system. You can either install the files manually or use the automated installation script provided.

The automated installation uses FTP to run an installation script (**install.ftp**) that allocates and sends the two transmit (.xmit) files using the names defined in the script. It then submits the **install.jcl** file to complete the installation.

You must modify **install.ftp** and **install.jcl** for your environment before running the script.

Note: *If you changed any of the installation DSNAMES you need to edit these members and correct the names as needed. They assume that the high-level qualifier is COBJMS.SHIPPING. The LE level in the JCL file may not match your system. Modify the JCL file as needed. In addition, you might also need to edit the JOBCARD data.*

To use the automated MVS installation

1. Navigate to the directory where you extracted the eGate API Kit files (under **"Downloading the eGate API Kit to a Local Machine" on page 10**).
2. Modify the user name and password in **install.ftp** to your user name and password for the MVS system.
3. If necessary, modify the MVS filenames in **install.jcl** to meet your requirements. If you modify the file names, you must also modify the file names in **install.ftp** to match.
4. At a command prompt, run the following command:

   ```
   ftp -s:install.ftp <hostname>
   ```

   where <hostname> is the name of the machine on which you want to install the eGate API Kit for COBOL.
5. To verify the installation, log in to the MVS system and run one of the sample files as described in **"Working with the COBOL API Samples" on page 26**.

To install to the MVS system manually

Use these procedures if the automated FTP installation script cannot be used.

1. Navigate to the directory where you extracted the eGate API Kit files (under **"Downloading the eGate API Kit to a Local Machine" on page 10**).
2. Allocate the z/OS files.
   - Allocate COBJMS.SHIPPING.JCL.CNTL.XMIT as follows:

     ```
     DSORG=PS  RECFM=FB LRECL=80 BLKSIZE=3120 -  15 blocks primary
     ```
   - Allocate COBJMS.SHIPPING.LOADLIB.XMIT as follows:

     ```
     DSORG=PS  RECFM=FB LRECL=80 BLKSIZE=3120 -  5000 blocks primary
     ```
3. Upload the images using the following commands:

   ```
   FTP <hostname>
   logon
   binary
   put cobjms.shipping.jcl.cntl.xmit 'COBJMS.SHIPPING.JCL.CNTL.XMIT'
   put cobjms.shipping.loadlib.xmit 'COBJMS.SHIPPING.LOADLIB.XMIT'
   ```
4. To receive the images, do the following:
   A. From a TSO command prompt, enter the following:

      ```
      TSO RECEIVE INDATASET('COBJMS.SHIPPING.JCL.CNTL.XMIT')
      ```

   B. When prompted, enter the following:

      ```
      DA('COBJMS.SHIPPING.JCL.CNTL')
      TSO RECEIVE INDATASET('COBJMS.SHIPPING.LOADLIB.XMIT')
      ```

   C. When prompted, enter the following:

      ```
      DA('COBJMS.SHIPPING.LOADLIB')
      ```
5. To verify the installation, log in to the MVS system and run one of the sample files as described in **"Working with the COBOL API Samples" on page 26**.

Chapter 3

JMS and COBOL Implementation Overview

The eGate API Kit provides an interface for external MVS applications to exchange data with Java CAPS Projects via the Sun SeeBeyond JMS IQ Manager. This chapter provides implementation information and 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 runtime Project.

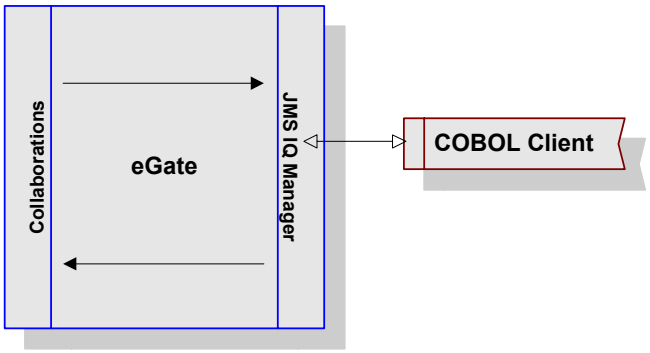
3.1.1 The Java CAPS JMS Interface

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.

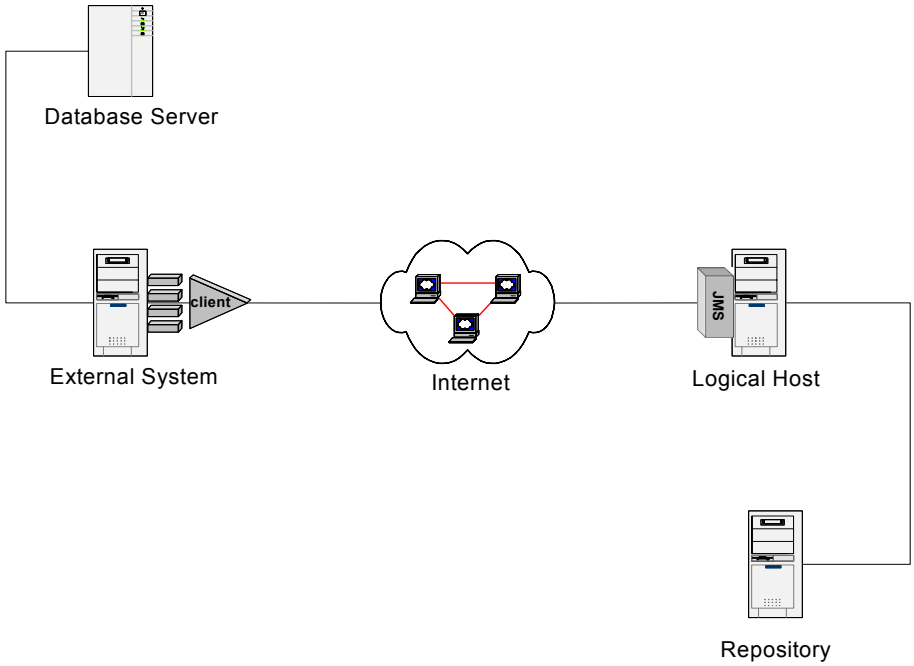
Figure 1 illustrates the communication between each component.

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



The following components are required:

- Repository
- Logical Host
- Sun SeeBeyond Integration Server (IS)
- 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, and the host name and port number of the JMS IQ Manager (see **"Viewing JMS IQ Manager Port Numbers"** for port number information).

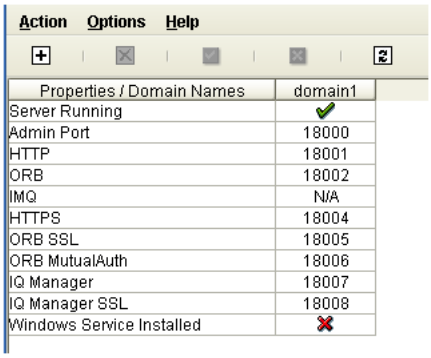
3.1.3 Viewing JMS IQ Manager Port Numbers

The default port number for Sun SeeBeyond 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** property specifies the JMS IQ Manager port; the **IQ Manager SSL** property specifies the JMS IQ Manager SSL port.

3.2 About the Java Messaging Service

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

3.2.1 JMS Messages

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

Note: *Not all the header fields described below are available through the COBOL API.*

- **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 JMS message ID 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 JMS message ID must start with the **ID:** prefix.
- **JMSTimestamp** - The time that a message is handed off to a provider to be sent. It is not the actual transmission time because the send may occur later due to pending transactions.
- **JMSExpiration** - The time that is calculated as the sum of the time-to-live value specified in the send function and the current GMT value. After the send method is returned, the message's JMSExpiration header field contains this value. If the time-to-live is specified as zero, expiration is also set to zero and the message does not expire.
- **JMSRedelivered** - An indicator of whether the message was re-delivered to the consumer. If the header is "true", the message is re-delivered; if the header is false, the message is not. The message might be marked as re-delivered if a consumer fails to acknowledge delivery of the message, or if the JMS provider is uncertain that the consumer received the message.

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

Message Properties

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

Message Body (Payload)

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

- **Text Message** - A message that carries a string (of type **java.lang.String**) as its payload, and is used for exchanging both text messages and XML documents. As such, it is the most frequently used message type.
- **Bytes Message** - A message that carries a byte array as its payload, and is often used in cases where JMS is simply used as a transport between systems. This message type is for encoding a body to match an existing message format. It can be used for exchanging data in an application's native format.

3.2.2 JMS Messaging Types

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

- **Request/Reply Messaging** on page 16
- **Send-only Messaging** on page 17
- **Receive-only Messaging** on page 17

Request/Reply Messaging

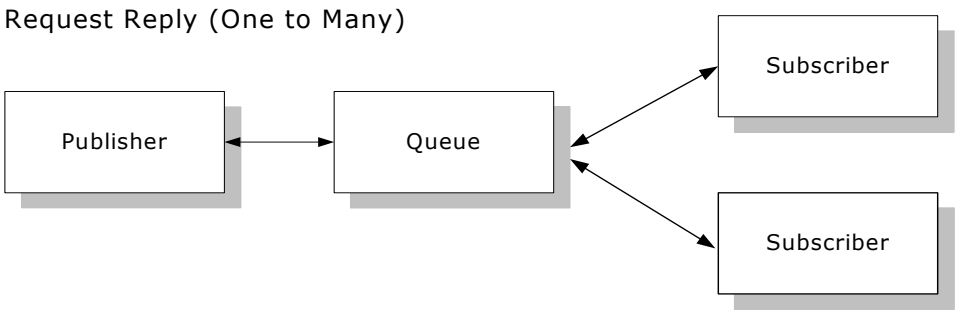
In request/reply messaging, data is sent to a Queue in the JMS server and a response is returned as follows:

1. A COBOL client submits data (the request) to the JMS server.
2. The data is processed as defined in the Java CAPS Project.

3 The JMS server returns data (the reply) to the same client application that submitted the request.

Note: *In request/reply messaging, the destination must be a Queue, not a Topic.*

Figure 4 Request/Reply Messaging



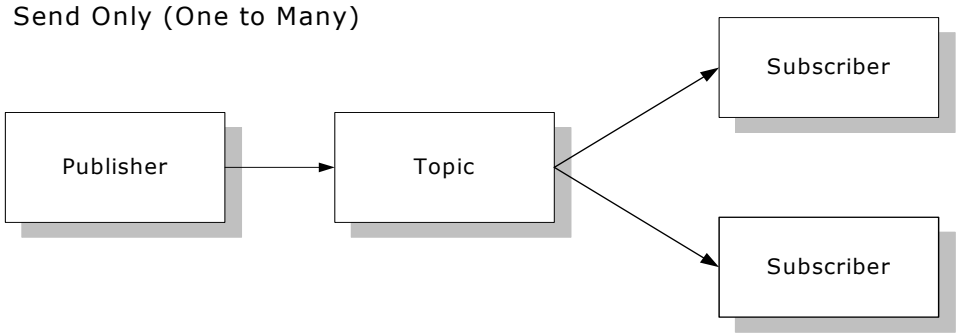
JMS provides the **JMSReplyTo** message header field for specifying the destination to which the reply to a message is to be sent. The **JMSCorrelationID** header field of the reply can be used to reference the original request.

Send-only Messaging

In send-only messaging, a message producer distributes a message to one or more consumers. Messages are published via a Topic or Queue without requiring a request. When a message is sent to a Topic, each subscriber to the Topic receives a copy of the message. Multiple subscribers can receive messages published by one producer. When a message is sent to a Queue, the message is only delivered to one of the Queue receivers.

Figure 5 illustrates send-only messaging using a Topic as the destination.

Figure 5 Send-only Messaging Using a Topic



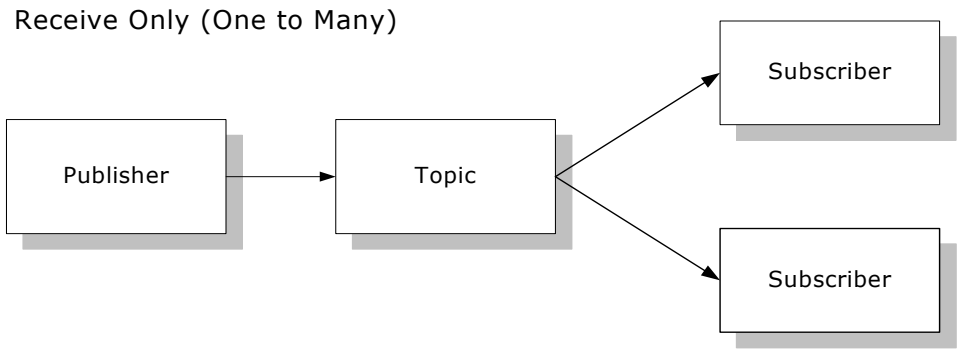
Receive-only Messaging

In receive-only messaging, unsolicited messages are received from an external system. Messages are received from either a Topic or Queue without requiring a request. When

the message is received from a Topic, each subscriber to the Topic receives a copy of the message. When the message comes from a Queue, the message is only delivered to one of the Queue receivers.

Figure 6 illustrates receive-only messaging when the subscriber receives from a Topic. In this case, several subscribers can receive the same message.

Figure 6 Receive-only Messaging Using a Topic



Chapter 4

COBOL Reference

The eGate API Kit provides COBOL functions to perform message transfers using a JMS Topic or Queue. This chapter describes the JMS functions you can use in your COBOL applications.

What's in This Chapter

4.1 COBOL API Overview

The eGate API Kit for COBOL supplies a set of JMS wrapper functions for COBOL to allow your applications to quickly access the Java CAPS JMS without your having to understand all JMS details. The API provides functions to allow you to perform standard JMS messaging scenarios, including request/reply, send-only, and receive-only.

A typical send sequence would be:

- Create a connection to the JMS server (ICNOPEN).
- Send a message to a Topic or Queue (ICNSEND).
- Close the connection (ICNCLOSE).

A typical request/reply sequence would be:

- Create a connection to the JMS server (ICNOPEN).
- Send a message to a Topic or Queue and receive a reply (ICNRQRP).
- Close the connection (ICNCLOSE).

This chapter provides a complete reference for the functions of the eGate API Kit for COBOL. **ICNAPI** provides additional information about the required values for the COBOL functions, and defines the error codes returned by each function.

4.2 COBOL Functions

The COBOL API includes the following functions:

ICNOPEN

Syntax

```
call "ICNOPEN" using by reference
     ICNAPI-handle
     ICNAPI-host
     ICNAPI-remote-port
     ICNAPI-dest-type
     ICNAPI-dest-name
     ICNAPI-errno
     ICNAPI-retcode
```

Description

ICNOPEN creates a connection to the JMS server running on the specified remote host and TCP/IP port. This socket connection is defined by a unique identifier, or handle, that is returned by the ICNOPEN function. This allows multiple connections to be opened and maintained by a single CICS application to one or more JMS servers.

Parameters

Parameter	Data Type	Description	Required
ICNAPI-handle	pic s9(8) binary	The returned handle of the open session	Yes, initialized to 0.
ICNAPI-host	pic x(24)	A 24-byte character field containing the DNS name or IP address of the remote host on which the JMS server is listening. Use “move Z” to populate this parameter. "	Yes
ICNAPI-remote-port	pic 9(8) binary	The JMS server TCP/IP port number.	Yes
ICNAPI-dest-type	pic 9(8) binary	The type of JMS destination being used, either QUEUE or TOPIC. See ICNAPI for values.	Yes
ICNAPI-dest-name	pic x(24)	The name of the Queue or Topic. Use “move Z” to populate this parameter.	Yes

Parameter	Data Type	Description	Required
ICNAPI-errno	pic 9(8) binary	Describes the cause of the error. Setting this value to "999" enables full debug logging for this session.	Yes, should be initialized to 0.
ICNAPI-retcode	pic 9(8) binary	The return code of the call. A value of zero or greater indicates a successful call. A negative value signifies an error. ERROR return values are defined in ICNAPI as 88 levels.	Yes, should be initialized to 0.

Return Values

The TCP/IP socket number for the established connection and a return code.

If the return code (**ICNAPI-retcode**) is zero (0) or greater, the function was successful. If the return code is negative, an error occurred and the error number is also returned in **ICNAPI-errno**.

ICNSEND

Syntax

```
call "ICNSEND" using by reference
     ICNAPI-handle
     ICNAPI-message
     ICNAPI-message-type
     ICNAPI-message-len
     ICNAPI-secs-to-expire
     ICNAPI-errno
     ICNAPI-retcode
```

Description

ICNSEND sends a message or block of data to the JMS server. The function then waits a specified time (expressed in hundredths of seconds) for an acknowledgment to arrive on the socket connection identified by the session handle.

Parameters

Parameter	Data Type	Description	Required
ICNAPI-handle	pic s9(8) binary	The handle of the open session.	Yes
ICNAPI-message	pic x(???)	Up to 8 MB containing the actual data to be sent. The contents of this field is transmitted without any conversion. Use "move Z" to populate the value of this parameter.	Yes
ICNAPI-message-type	pic 9(8) binary	The type of message being sent. Specify either TEXT or BINARY.	Yes

Parameter	Data Type	Description	Required
ICNAPI- message-len	pic 9(8) binary	The length in bytes of the message body.	Yes
ICNAPI-secs-to-expire	pic 9(8) binary	The time (in hundredths of a second) to wait for a send acknowledgement.	No
ICNAPI-errno	pic 9(8) binary	Describes the cause of the error. Setting this value to "999" enables full debug logging for this session.	Yes, should be initialized to 0.
ICNAPI-retcode	pic 9(8) binary	The return code of the call. A value of zero or greater indicates a successful call. A negative value signifies an error. ERROR return values are defined in ICNAPI as 88 levels.	Yes, should be initialized to 0.

Return Values

A return code indicating the success of the transmission.

If the return code (**ICNAPI-retcode**) is zero (0) or greater, the function was successful. If the return code is negative, an error occurred and the error number is also returned in **ICNAPI-errno**.

ICNRECV

Syntax

```
call "ICNRECV" using by reference
     ICNAPI-handle
     ICNAPI-returnmsg
     ICNAPI-message-type
     ICNAPI-returnmsg-len
     ICNAPI-hsecs-to-wait
     ICNAPI-errno
     ICNAPI-retcode
```

Description

ICNRECV receives a message or block of data from the JMS server. The function waits a specified time (expressed in hundredths of seconds) for a message to arrive on the socket connection identified by the session handle.

Parameters

Parameter	Data Type	Description	Required
ICNAPI-handle	pic s9(8) binary	The handle of the open session.	Yes

Parameter	Data Type	Description	Required
ICNAPI-returnmsg	pic X(???)	Up to 8 MB containing the actual data being received. The contents of this field is transmitted without any conversion. Use “move Z” to populate the value of this parameter.	Yes (returned_
ICNAPI-returnmsg-len	pic 9(8) binary	The length in bytes of the message body.	Yes (returned)
ICNAPI-message-type	pic 9(8) binary	The type of message being received, either TEXT or BINARY.	Yes (returned)
ICNAPI-hsecs-to-wait	pic 9(8) binary	The time (in hundredths of a second) to wait for a response from the server.	No
ICNAPI-errno	pic 9(8) binary	Describes the cause of the error. Setting this value to “999” enables full debug logging for this session.	Yes, should be initialized to 0.
ICNAPI-retcode	pic 9(8) binary	The return code of the call. A value of zero or greater indicates a successful call. A negative value signifies an error. ERROR return values are defined in ICNAPI as 88 levels.	Yes, should be initialized to 0.

Return Values

The length, in bytes, of the data received from the JMS server along with the message data.

If the return code (**ICNAPI-retcode**) is zero (0) or greater, the function was successful. If the return code is negative, an error occurred and the error number is also returned in **ICNAPI-errno**.

ICNRQRP

Syntax

```
call "ICNRQRP" using by reference
     ICNAPI-handle
     ICNAPI-message
     ICNAPI-message-type
     ICNAPI-message-len
     ICNAPI-returnmsg
     ICNAPI-message-type
     ICNAPI-returnmsg-len
     ICNAPI-hsecs-to-wait
     ICNAPI-errno
     ICNAPI-retcode
```

Description

ICNRQRP receives a message or block of data from the JMS server. The function waits a specified time (expressed in hundredths of seconds) for a message to arrive on the socket connection identified by the session handle. The outbound message can only be sent to a Queue.

Parameters

Parameter	Data Type	Description	Required
ICNAPI-handle	pic s9(8) binary	The handle of the open session.	Yes
ICNAPI-message	pic X(???)	Up to 8 MB containing the actual data to be sent. The contents of this field is transmitted without any conversion. Use "move Z" to populate the value of this parameter.	Yes
ICNAPI-message-type	pic 9(8) binary	The type of message being sent. Specify either TEXT or BINARY	
ICNAPI-message-len	pic 9(8) binary	The length in bytes of the body of the sent message.	
ICNAPI-returnmsg	pic X(???)	The message body of the reply.	Yes
ICNAPI-message-type	pic 9(8) binary	The type of message returned, either TEXT or BINARY	
ICNAPI-returnmsg-len	pic 9(8) binary	The length of the body of the returned message.	
ICNAPI-hsecs-to-wait	pic 9(8) binary	The number seconds to wait for a send or request/reply acknowledgement.	No
ICNAPI-errno	pic 9(8) binary	Describes the cause of the error. Setting this value to "999" enables full debug logging for this session.	Yes, should be initialized to 0.
ICNAPI-retcode	pic 9(8) binary	The return code of the call. A value of zero or greater indicates a successful call. A negative value signifies an error. ERROR return values are defined in ICNAPI as 88 levels.	Yes, should be initialized to 0.

Return Values

The length, in bytes, of the data received from the JMS server along with the message data.

If the return code (**ICNAPI-retcode**) is zero (0) or greater, the function was successful. If the return code is negative, an error occurred and the error number is also returned in **ICNAPI-errno**.

ICNCLOSE

Syntax

```
call "ICNCLOSE" using
     ICNAPI-handle
     ICNAPI-errno
     ICNAPI-retcode
```

Description

ICNCLOSE shuts down the socket connection with the JMS server and frees any associated resources.

Parameters

Parameter	Data Type	Description	Required
ICNAPI-handle	pic s9(8) binary	The handle of the open session.	Yes
ICNAPI-errno	pic 9(8) binary	Describes the cause of the error. Setting this value to “999” enables full debug logging for this session.	Yes, should be initialized to 0.
ICNAPI-retcode	pic 9(8) binary	The return code of the call. A value of zero or greater indicates a successful call. A negative value signifies an error. ERROR return values are defined in ICNAPI as 88 levels.	Yes, should be initialized to 0.

Return Values

A return code indicating the success of the transmission.

If the return code (**ICNAPI-retcode**) is zero (0) or greater, the function was successful. If the return code is negative, an error occurred and the error number is also returned in **ICNAPI-errno**.

Chapter 5

Working with the COBOL API Samples

The eGate API Kit for JMS IQ Manager includes sample Java CAPS Projects along with COBOL code samples so you can get started quickly. This chapter describes how to run the code samples to transfer messages between COBOL applications the JMS Server.

What's in This Chapter

5.1 About the COBOL Samples

The eGate API Kit provides sample Java CAPS Projects and COBOL code samples designed to work together to demonstrate sending and receiving messages between a Sun SeeBeyond JMS server and a COBOL client running on MVS.

The COBOL code samples are included in the file you downloaded when you installed the COBOL API Kit (**"Downloading the eGate API Kit to a Local Machine" on page 10**). There are four COBOL samples:

- **ICNBTEST** is a batch sample that sends a message to a Queue.
- **JMSRQRP** is a batch sample that defines a request/reply scenario. This sample can be run with the JMSRQRP505 sample Java CAPS Project.
- **ICNBQ2Q** is a batch sample that defines a send and receive scenario. This sample can be run with the JMSQ2Q505 sample Java CAPS Project.
- **JMSCTEST** is a CICS sample that can perform a simple send and receive transaction and also defines a request/reply scenario to use with the JMSRQRP505 sample Java CAPS Project.

You can access the Project samples from the **Documentation** page of the Suite Installer. The eGate API Kit sample file, **eGateAPIKit_Sample.zip**, contains the Project samples for all kits in an embedded .zip file named **ProjectSamples.zip**. The Project samples are described in **"Implementing the Java CAPS Projects"** on the following page.

5.2 Implementing the Java CAPS Projects

The eGate API Kit samples includes two Java CAPS Projects so you can get quickly started using the sample COBOL programs with the Sun SeeBeyond JMS IQ Manager. Both Projects include sample Environments you can use, but you need to configure them for your system.

The samples are provided in the following files located in **ProjectSamples.zip**:

- **JMSRQRP505.zip** - This sample includes a Project named **JMSRQRP505** with an Environment named **EnvCOBRQRP**. This is a simple JMS Project designed for request/reply scenarios. Use this sample with the JMSRQRP and JMSCTEST COBOL samples.
- **COBJMSQ2Q505.zip** - This sample includes a Project named **JMSQ2Q505** with an Environment named **Environment1**. This is a simple JMS Project that includes an inbound Queue that sends data to an outbound Queue via a Java Collaboration. Use this sample with the ICNBQ2Q COBOL sample.

Before continuing, make sure you have downloaded the sample file as described in **"Downloading the eGate API Kit to a Local Machine" on page 10**. Implementing the sample Projects consists of the following steps.

- **Importing the Sample Projects** on page 27
- **Creating the Environment** on page 28
- **Deploying the Project** on page 28

5.2.1 Importing the Sample Projects

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

Note: *The Projects are compatible with Java CAPS versions 5.0.5 and 5.1.3.*

To import the sample Project into Enterprise Designer

1. If you have not already done so, extract **eGateAPIKit_Sample.zip**.
2. Navigate to the directory where you extracted the sample files, and then extract **ProjectSamples.zip**.
3. Start Enterprise Designer.
4. From the Repository context menu, select **Import Project**.
5. A message box appears, prompting you to save any 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**.
6. Click the **Browse** button.

7 Do one of the following:

- To import the Project that works with the JMSRQRP COBOL sample, locate and select **JMSRQRP505.zip**, located in the directory in which you extracted **ProjectSamples.zip**.
- To import the Project that works with the ICNBQ2Q COBOL sample, locate and select **COBJMSQ2Q505.zip**, located in the directory in which you extracted **ProjectSamples.zip**.

8 Click **Open**.

The Import Manager dialog appears.

9 Click **Import**.

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.

10 Click **OK**.

11 When you are finished importing files, click **Close**. The Project Explorer is automatically refreshed from the Repository.

5.2.2 Creating the Environment

When you import one of the sample Projects, the associated Environment is also imported. You can use the sample or create a new Environment. Use the Environment Explorer of Enterprise Designer to configure the existing Environment or to create a new Environment and Logical Host.

The Logical Host must include a Sun SeeBeyond JMS IQ Manager and Integration Server (IS). Modify the properties for the IS by specifying the login information. Modify the JMS IQ Manager by specifying the URL and login information.

For more information about Environments, see the *Sun SeeBeyond eGate Integrator User's Guide* or the *Sun SeeBeyond eGate Integrator System Administration Guide*.

5.2.3 Deploying the Project

For each Project you imported, you need to 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 Project, make sure the Logical Host for the sample applications is started. For more information about Deployment Profiles and Logical Hosts, see the *Sun SeeBeyond eGate Integrator User's Guide* or the *Sun SeeBeyond eGate Integrator System Administration Guide*.

Important: *You will need to modify the host name in the COBOL source code to match the server to which the Project is deployed and the port number to match the JMS server*

port number (for information about port numbers, see **Viewing JMS IQ Manager Port Numbers** *on page 14).*

5.3 Modifying the COBOL Source Code

For each of the COBOL samples provides, you need to modify the source files by changing the parameters described in Table 2 so they connect to an existing JMS IQ Manager server. Review the files for any other changes that might be required to suit your environment.

Table 2 COBOL JCL Parameters

Parameter	Description
ICNAPI-host	The name of the machine on which the Sun SeeBeyond JMS Server resides.
ICNAPI-remote-port	The port number on which the JMS server is listening.
ICNAPI-dest-name	The name of the Queue to which a message will be sent or from which a message will be received.

In addition, you might need to modify the sample JCL files used to run each job. Before you run the samples, review the JCL file used to be sure it matches your system configuration. To reduce the need for system configuration changes, use all uppercase characters in the Queue names for the CICS sample.

5.4 Sample JCL to Link, Compile, and Run Jobs

Below is a sample JCL file to compile, link, and run the sample code. This sample is designed for the ICNBTEST batch sample.

Note: *Your installation may require changes to conform to your standards.*

```
//SAMPLE1  JOB    (),CLASS=A,MSGCLASS=X,                          00001012
//         MSGLEVEL=(1,1),NOTIFY=&SYSUID,TIME=1440                00004003
//  SET LEPFX='CEE.V2R1M0'                                        00007019
//COMPCOB EXEC,IGYWCPL,REGION=80M,GOPGM=ICNBTEST,,                00039017
//        PGMLIB=COBJMS.SHIPPING.LOADLIB,                         00039118
//        PARM.COBOL='RENT,PGMNAME(LONGMIXED)'                    00039311
//COBOL.SYSIN   DD DSN=COBJMS.SOURCE.COBOL(ICNBTEST),DISP=SHR     00039514
//PLKED.SYSIN   DD                                                00039605
//              DD DSN=COBJMS.SOURCE.COBOL(MVSICANX),DISP=SHR     00039713
//LKED.SYSIN    DD DSN=COBJMS.LOADLIB(MVSICAN),DISP=SHR           00039813
//*--------------------------------------------------------------- 00040204
//* EXECUTE THE MAIN PROGRAM, LOADING MVSICAN DLL..               00040313
//*--------------------------------------------------------------- 00040404
//STEP3 EXEC PGM=ICNBTEST,REGION=80M                              00040515
//STEPLIB DD DSN=COBJMS.LOADLIB,DISP=SHR                          00040612
//SYSOUT  DD SYSOUT=*                                             00040704
//CEEDUMP DD SYSOUT=*                                             00040804
```

5.5 Running the CICS Sample

The CICS sample consists of one program (JMSCTEST) and three BMS maps. It also supplies the source JCL to build the maps (GENMAPJ) and the JCL to compile the sample (COBJMS1). You need to modify the JCL for your environment.

The CICS sample sends a message to and receives a message from a Queue in a Java CAPS Project. It returns a code indicating whether the message transmission was successful. The sample sends a request to a Queue and receives the socket number of the connection in response.

You can use the JMSRQRP505 sample Project with this program when running the request/reply scenario or you can create a new Project. To run the send and receive scenario, you need a running JMS server with a Queue to receive the message. As supplied, the sample links into the COBJMS.SHIPPING.LOADLIB dataset.

Refer to the comments in the sample program for ideas on how to use the API.

To run the sample application

1. Add the COBJMS.SHIPPING.LOADLIB dataset to the concatenation for DFHRPL in the CICS startup.
2. Run GENMAPJ and COBJMS1 to build the maps and program.
3. Run the CEDAJMS job to add the MAP, PROG, and TRANS definitions to the CICS tables.
4. Run `CEMT set prog() newcopy ena` for each of the following:
   - JMSCICS
   - JMSEMAP
   - JMSOMAP
   - JMSSMAP
   - MSAPI
   - MSCLI
   - MSCOM
   - MVSICAN
5. Run the job JMSCICS1 to compile, link, and run the CICS sample.
6. The transaction is JMSO. At the JMSO prompt, enter the host name and port number of the JMS server and the name of the configured Queue.

   If the connection is successful, the return code is 0 (zero).
7. To send and receive a message:
   A. When prompted to send, receive, or quit, select **Send Msg** (PF5), which sends one message to the JMS Queue.
   B. Select **Recv** (PF6) to receive the message back from the same Queue.

8 To perform a request/reply transaction, select **Req-Repl** (PF7). This uses the JMSRQRP505 Project sample, and returns the socket number as the reply.

5.6 Implementing the Batch Samples

There are three batch samples provided with the eGate API Kit for COBOL. You can modify and use any of these samples to learn more about how to implement the COBOL functions. To work with the samples, perform any of the following procedures:

- **Running the ICNBTEST Sample** on page 31
- **Running the JMSRQRP Sample** on page 31
- **Running the ICNBQ2Q Sample** on page 32

The error code parameter for each of the batch samples is set to "999" to provide a complete information about the transactions. When you complete a sample job, check the output using System Display and Search Facility (SDSF).

Refer to the comments in the sample programs for ideas on how to use the API.

5.6.1 Running the ICNBTEST Sample

The ICNBTEST sample program sends a simple message to a Queue using a Collaboration in the sample Project to transfer the message. As supplied, the sample links into the COBJMS.SHIPPING.LOADLIB dataset. You need to modify the source code; otherwise the sample will result in a timeout being posted in the job log since the predefined host, port, and Queue do not exist.

To run the ICNBTEST sample

1 Modify the source file, **ICNBTEST**, as described in **"Modifying the COBOL Source Code" on page 29**.

2 Add the COBJMS.SHIPPING.LOADLIB dataset to the concatenation for DFHRPL in the CICS startup.

3 Do one of the following:

- Run the job LICNBAT to compile, link, and run the ICNBTEST sample.
- Run the job RUNBAT to run the ICNBTEST sample without compiling or linking.

5.6.2 Running the JMSRQRP Sample

The JMSRQRP sample program sends a request to a Queue named "COBIN" in the JMSRQRP505 sample Project and then receives a response. As supplied, the sample links into the COBJMS.SHIPPING.LOADLIB dataset. You need to modify the source code; otherwise the sample will result in a timeout being posted in the job log since the predefined host, port, and Queue do not exist.

To run the JMSRQRP sample

1. Modify the source file, **JMSRQRP**, as described in **"Modifying the COBOL Source Code" on page 29**.
2. Add the COBJMS.SHIPPING.LOADLIB dataset to the concatenation for DFHRPL in the CICS startup.
3. Run the job LICNBAT2 to compile, link, and run the JMSRQRP sample.

5.6.3 Running the ICNBQ2Q Sample

The ICNBQ2Q sample program sends a messages to a Queue named "COBIN" and receives a message from a Queue named "COBOUT" in the sample Project JMSQ2Q505. As supplied, the sample links into the COBJMS.SHIPPING.LOADLIB dataset. You need to modify the source code; otherwise the sample will result in a timeout being posted in the job log since the predefined host, port, and Queue do not exist.

To run the ICNBQ2Q sample

1. Modify the source file, **ICNBQ2Q**, as described in **"Modifying the COBOL Source Code" on page 29**.
2. Add the COBJMS.SHIPPING.LOADLIB dataset to the concatenation for DFHRPL in the CICS startup.
3. Run the job LICNBAT3 to compile, link, and run the ICNBQ2Q sample.

Index

P

Q

R

S

T