SUN SEEBEYOND

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

Release 5.1.2

Sun microsystems

Part Number: 819-7432-10

Version 20060929201546

Contents

Chapter 1

Introduction

This chapter introduces you to this document, 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 document describes how to install and use the eGate™ API Kit to create Java 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" on page 8** describes how to install the eGate API Kit and its samples.
- **Chapter 3 "JMS and Java Implementation Overview" on page 11** describes how to develop Java applications to access the Sun SeeBeyond JMS IQ Manager in Java CAPS Projects.
- **Chapter 4 "Working with the Java API Samples" on page 21** describes how to build and run the Java sample included in the eGate API Kit.

1.1.2 Intended Audience

This document is intended for developers who are familiar with programming Java 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, refer to the following documents:

- *Sun SeeBeyond eGate Integrator JMS Reference Guide*
- *Sun SeeBeyond Java Composite Application Platform Suite Installation Guide*
- *Sun SeeBeyond eGate Integrator User's 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 the JDK™ software, version 1.4.

2.3 Supported Compilers

When compiling your Java JMS client applications, you can use any standard Java compiler that is compliant with JDK version 1.4.

2.4 Installing the eGate API Kit

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

Before you install 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 eGate API Kit:

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

To install 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_Java** (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 and click **Next** after each SAR file is selected:
   - **eGate_APIKit_Java.sar**.
   - **eGateAPIKitDocs.sar**

   When the installation is finished, the "Installation Completed" message appears.
5. In the **Downloads** page, select one of the following items, select a location for the **.zip** file to be saved, and then extract the file.
   - **API kit Java (zip file)** - for Windows installations
   - **API kit Java (tar file)** - for UNIX installations
6. To download the sample Projects and code samples, select 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 and working with the sample code, see **Chapter 4 "Working with the Java API Samples"**.

2.5 Post-Installation Instructions

After the eGate API Kit installation, do the following:

1. Locate the **com.stc.jms.jmsis.jar, jta.jar, jms.jar,** and **log4j.jar** files in the folder where you installed the eGate API Kit (these are the file extracted from the **API kit Java** file).
2. Modify the **CLASSPATH** to include the **jms.jar**, **com.stc.jmsis.jar**, (for JNDI connections) and **log4j.jar** (when connecting to JNDI through the Information Server) files.
3. For XA support, include the **jta.jar** in your path.
4. For HP-UX, make sure that the JVM has been started with the `-XdoCloseWithReadPending` flag, for example:

```
java -XdoCloseWithReadPending -cp com.stc.jmsis.jar;.;jms.jar
Sample
```

Chapter 3

JMS and Java Implementation Overview

The eGate API Kit provides an interface for external applications to exchange data with Java CAPS Projects via the Sun SeeBeyond JMS IQ Manager. This chapter gives an overview of the JMS IQ Manager along with considerations to keep in mind when developing external applications to work with runtime Java CAPS Projects.

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 things to consider for the Java CAPS Project. This section also describes how to find out the port numbers used for a particular runtime Project.

3.1.1 JMS Specification

The Java Edition of the eGate API Kit supports the *Java Message Service Specification version 1.1*. For a complete reference of the Java classes and methods available to the eGate API Kit, see the *Java™ Message Service (JMS) API Documentation version 1.1*.

3.1.2 The Java CAPS JMS Interface

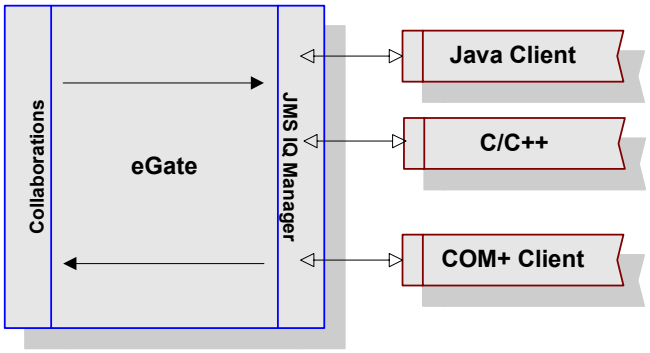
For those of you unfamiliar with the Java CAPS JMS interfaces, this section provides an overview. The Java CAPS JMS consists of the following components:

- **Message Service Client** - The external application

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

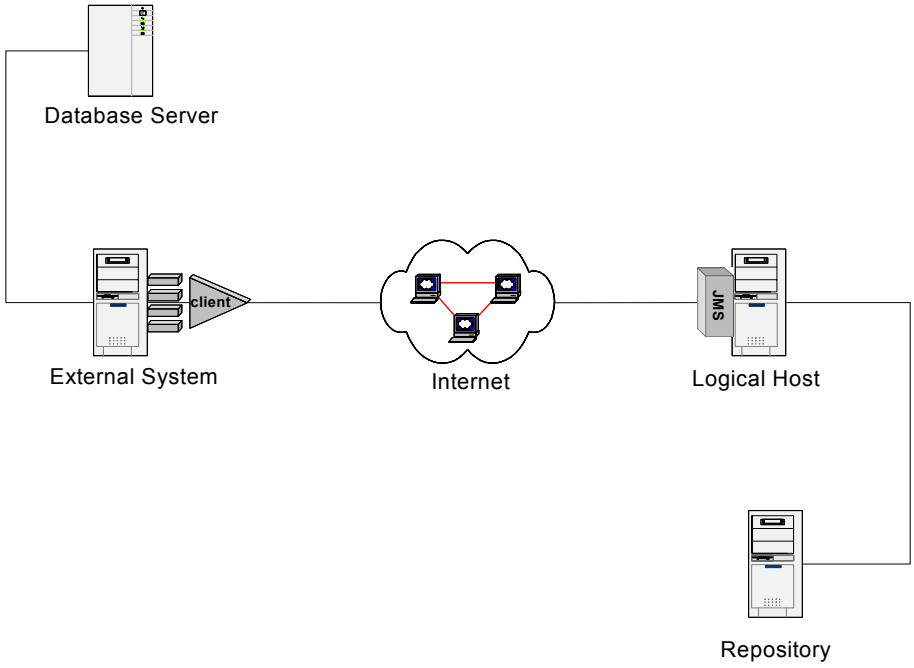
Figure 1 illustrates the communication between each component.

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.*

3.1.3 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. For instructions for displaying a JMS IQ Manager's port numbers, refer to **"Viewing JMS IQ Manager Port Numbers" on page 13**.
- The methods used must correspond to the expected data format.

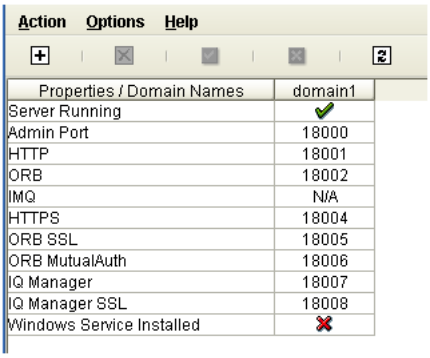
3.1.4 Viewing JMS IQ Manager Port Numbers

The default port number for JMS IQ Manager 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.

3.1.5 Sample Code

The eGate API Toolkit provides a sample download that includes code samples for creating interfaces to the Java CAPS JMS using Java. For information about implementing the sample code, see **"Working with the Java API Samples" on page 21**.

3.2 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.

3.3 Instantiating ConnectionFactories

To create connection factories for JMS IQ Manager directly, create instances of one of the following classes:

- com.stc.jms.client.STCQueueConnectionFactory
- com.stc.jms.client.STCTopicConnectionFactory
- com.stc.jms.client.STCXAQueueConnectionFactory
- com.stc.jms.client.STCXATopicConnectionFactory

These classes come with several different constructors. The two most useful constructors are:

```
public STCConnectionFactory(String host, int port)
```

and

```
public STCConnectionFactory(Properties p)
```

The latter constructor allows a number of advanced properties to be set. These properties are described below.

com.stc.jms.sockets.ServerHost

Specifies the host name of the machine the message server is running on.

com.stc.jms.sockets.ServerPort

Specifies the port number that the message server is listening on.

com.stc.jms.sockets.ConnectionManager.class

Specifies which threading model will be used for asynchronous communication. Valid values are "com.stc.jms.sockets.MuxConnectionMgr" and "com.stc.jms.sockets.ThreadPerConnectionMgr" (default). The former will use one thread for all socket connections; the second will use one thread for each socket. Note that there will be one socket in use for each consumer; when consumers are used with a message listener, or are used with a receive(timeout), asynchronous

communication is used. Using the "MUX"-model reduces the number of threads, but is more CPU intensive because there will be one thread polling sockets for incoming data.

com.stc.jms.sessionpooling

Specifies whether session pooling is used. Valid values are true and false (default). When session pooling is used, sessions are not physically closed when the close() method on a session is called. Instead they are returned to a pool; when a new session needs to be created, an attempt is made to reuse a session from the pool. Because a separate socket connection is used for each session, session pooling will speed up performance in situations where the application is creating and closing sessions rapidly. Note that sessions are pooled per connection, i.e. the pool of sessions is not global but each connection has its own session pool. When sessions are pooled, producers are also pooled automatically, i.e. when the close() method on a producer is called, the producer is not closed but is returned to the session's producer pool. When a new producer is created, an attempt is made to satisfy this request using the producer pool. Because each producer uses a socket connection, producer pooling improves performance in applications that rapidly create and close producers. Sessions and producers are closed physically, when the JMS connection that created them is closed physically.

com.stc.jms.connectionpooling

Specifies whether connection pooling is used. Valid values are true and false (default). When connection pooling is used, a connection is not physically closed when the close() method on a connection is called. Instead, the connection is returned to a global pool. Connection pooling is useful only in combination with session pooling.

com.stc.jms.connectionpooling.maxidlesec

Specifies the number of seconds that idle JMS connections will reside in the global connection pool before they are closed by a subsequent connection request. The default is 30 seconds.

com.stc.jms.ssl.authenticationmode

Determines if TCP/IP connections made by the client to the server will use SSL, and if so, how the server will be authenticated. Valid values are Authenticate, TrustAll and False (default). If no value is specified, or if False is specified, connections will not use SSL. A value of Authenticate will cause the client to assure that the certificate presented by the JMS server is trusted, i.e. is signed by an authority present in the truststore. See below on how to specify the truststore. A value of TrustAll will cause the client to accept any certificate presented by the JMS server. Note: if there is no value specified for this property, it will be looked up in the System properties.

javax.net.ssl.trustStore

Specifies the path to the store that contains trusted certificate authorities. This store is used to authenticate the JMS server when using SSL and the Authenticate authentication mode. Note: if there is no value specified for this property, it will be looked up in the System properties.

javax.net.ssl.trustStorePassword

Specifies the password used to protect the certificate store that contains trusted certificate authorities. This store is used to authenticate the JMS server when using SSL and the Authenticate authentication mode. Note: if there is no value specified for this property, it will be looked up in the System properties.

security.provider.1

Specifies the security provider used for SSL (default: com.sun.net.ssl.internal.ssl.Provider). Note that if the provider is set to com.ibm.jsse.IBMJSSEProvider, the IbmX509 TrustManagerFactory will be used; in other cases, the SunX509 TrustManagerFactory will be used. Note: if there is no value specified for this property, it will be looked up in the System properties.

3.4 Using the Built-In JNDI Provider

The JMS IQ Manager can be made to look like a JNDI server, that is, the server can appear as if it has a JNDI server built in.

How it works

The `com.stc.jms.stcjms.jar` now has a few classes that implement the `javax.naming.Context` and related classes. When an application instantiates the initial context and does a lookup, these classes will connect to the server and retrieve a list of topics and queues. The client converts the strings retrieved from the server into `STCTopic`-s and `STCQueue`-s and. These topics and queues then can be used in calls to `send(Queue)` etc. The JNDI provider also provides connection factories.

Practical use

This JNDI provider is available in the Integration Server. Clients can use this provider directly to lookup connection factories and destinations rather than connecting to the application server's JNDI server. This is useful in cases where normal JNDI lookup is difficult due to firewalls.

How to instantiate the initialcontext

Sample code:

```
Properties p = new Properties();
p.put(InitialContext.INITIAL_CONTEXT_FACTORY,
"com.stc.jms.jndispi.InitialContextFactory");
p.put(InitialContext.PROVIDER_URL, "stcms://localhost:18007");
p.put(InitialContext.SECURITY_PRINCIPAL, "Administrator");
p.put(InitialContext.SECURITY_CREDENTIALS,
TestConstants.getPassword());
InitialContext ctx = new InitialContext(p);
Topic t  = (Topic) ctx.lookup("topics/myTopic");
```

Hence, the following properties need to be defined as in the following example:

```
java.naming.factory.initial=com.stc.jms.jndispi.InitialContextFactory
java.naming.provider.url=stcms://localhost:18007
java.naming.security.principal=Administrator
java.naming.security.credentials=STC
```

The latter two are only needed if security is enabled for JMS IQ Manager.

The complete properties set is offered to the STCConnectionFactory. Hence, the properties above can be mixed with properties like

```
com.stc.jms.strictPersistence = true
```

This is exactly how the connection properties are assembled:

- use the properties as passed to `InitialContext`
- use the host and port number as specified in the `InitialContext.PROVIDER_URL`, possibly overwriting already defined properties
- merge additional properties specified in the query string of the `InitialContext.PROVIDER_URL`, possibly overwriting already defined properties

These connection properties are used both for connecting to the server to obtain the list of queues and topics, as well as to configure the connection factories that are bound in the JNDI tree.

The JNDI tree

The JNDI tree is built up as follows:

```
root
   topics
       topic1
       topic2
       etc
  queues
      queue1
      queue2
      etc.
  connectionfactories
      queueconnectionfactory
      topicconnectionfactory
      connectionfactory
      xaqueueconnectionfactory
      xatopicconnectionfactory
      xaconnectionfactory
```

Using the JNDI provider without connecting to the server

Typically, looking up a destination using `lookup`, for example, `lookup("queues/myqueue")` connects to the server and retrieves a list of queues. You can change this behavior so that the `lookup` method assumes the destination object is on the server, and just creates a queue or topic object on the client side **without connecting to the server**.

To select this behavior, specify this property in either the InitialContext, or in the connection URL:

```
com.stc.jms.jndispi.disconnected = true
```

Example

```
Properties p = new Properties();
p.put(InitialContext.INITIAL_CONTEXT_FACTORY,
"com.stc.jms.jndispi.InitialContextFactory");
p.put(InitialContext.PROVIDER_URL, "stcms://localhost:18007");
p.put(InitialContext.SECURITY_PRINCIPAL, "Administrator");
p.put(InitialContext.SECURITY_CREDENTIALS, "STC");
p.put("com.stc.jms.jndispi.disconnected", "true");
InitialContext ctx = new InitialContext(p);
```

```
Topic t  = (Topic) ctx.lookup("topics/myTopic");
```

which is equivalent to:

```
Properties p = new Properties();
p.put(InitialContext.INITIAL_CONTEXT_FACTORY,
"com.stc.jms.jndispi.InitialContextFactory");
p.put(InitialContext.PROVIDER_URL, "stcms://
localhost:18007?com.stc.jms.jndispi.disconnected=true"); <<<<<<<
p.put(InitialContext.SECURITY_PRINCIPAL, "Administrator");
p.put(InitialContext.SECURITY_CREDENTIALS,
TestConstants.getPassword());
InitialContext ctx = new InitialContext(p);
Topic t  = (Topic) ctx.lookup("topics/myTopic");
```

3.4.1 Registering JNDI Using the JNDIRegister Tool

To bind the ConnectionFactories from the command line:

The JMS IQ Manager also has functionality built in to create ConnectionFactories and bind them into any JNDI provider. This functionality can be invoked from the command line. The class com.stc.jms.client.JNDIRegister utility class is located in the com.stc.jmsis.jar. To use this utility do the following:

```
Usage: JNDIRegister -fact InitialCtxFactory -url ProviderUrl -type
Object type
      -jndiname JNDI name to be used to register the object
      [-dest Destination name] [-host host-name] [-port port]


Options/Arguments:
       InitialCtxFactory       *Initial context factory class, e.g.
com.sun
.jndi.fscontext.RefFSContextFactory
       ProviderUrl             *Url to access the JNDI provider URL,
e.g. file://stcms.jndi
       Object type             *Type: Q=queue connection factory;
q=queue;
                               T=topic connection factory; t=topic
       JNDI name to be used to register the object
                              *Name that the object will be registered
under
       -dest Destination name  Name of queue or topic
       -host host-name         specifies host name
       -port port              specifies server port number

* Required
```

To register a queue

```
java -cp com.stc.jmsis.jar;jms.jar;fscontext.jar;providerutil.jar
com.stc.jms.client.JNDIRegister
-fact com.sun.jndi.fscontext.RefFSContextFactory -url file://c:\temp
-type q -jndiname jndi-myqueue -dest myqueue
```

To register a topic

```
java -cp com.stc.jmsis.jar;jms.jar;fscontext.jar;providerutil.jar
com.stc.jms.client.JNDIRegister
-fact com.sun.jndi.fscontext.RefFSContextFactory -url file://c:\temp
-type t -jndiname jndi-mytopic -dest mytopic
```

To register a connection factory

```
java -cp com.stc.jmsis.jar;jms.jar;fscontext.jar;providerutil.jar
com.stc.jms.client.JNDIRegister
-fact com.sun.jndi.fscontext.RefFSContextFactory -url file://c:\temp
-type Q -jndiname jndi-myqfact -h 127.0.0.1 -p 7125
```

3.5 Implementing Message Selectors

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

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

For more information about Message Selection, see the *Java Message Service Specification Version 1.1* available at:

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

3.6 Implementing XA

XA compliance is achieved when cooperating software systems contain sufficient logic to ensure that the transfer of a single unit of data between those systems is neither lost nor duplicated because of a failure condition in one or more of the cooperating systems.

For more information on XA, see the *Sun SeeBeyond eGate Integrator User’s Guide*.

3.7 Implementing Secure Socket Layers (SSL)

A TCP/IP connection between a client system and a server can be secured by implementing Secure Socket Layers (SSL). The SeeBeyond Message Service listens on two ports at the same time. While the default port listens for non-SSL connections, the other can be restricted to only accept an SSL connection.

Connecting to the Message Server Using SSL:

The following sample shows how to connect to the message server using SSL by instantiating a connection factory directly.

```
import com.stc.jms.client.STCQueueConnectionFactory;
import javax.jms.*;
import java.util.*;
```

```
public class Sample {
  public static void main(String[] args) {
    try {
      Properties p = new Properties();
      p.put("com.stc.jms.sockets.ServerPort", "9225");
      p.put("com.stc.jms.sockets.ServerHost", "localhost");
      p.put("com.stc.jms.ssl.authenticationmode", "TrustAll");
      QueueConnectionFactory fact
        = new STCQueueConnectionFactory(p);
      QueueConnection conn = fact.createQueueConnection();
      QueueSession session = conn.createQueueSession(false,
        Session.AUTO_ACKNOWLEDGE);
      Queue dest = session.createQueue("Test");
      QueueSender producer = session.createSender(dest);
      TextMessage msg1 = session.createTextMessage("Hello world!");
      producer.send(msg1, DeliveryMode.NON_PERSISTENT,
        Message.DEFAULT_DELIVERY_MODE, 0);
      conn.close();
    } catch (JMSException ex) {
      ex.printStackTrace();
    }
  }
}
```

3.8 Logging

Java CAPS JMS clients use JDK logging by default rather than log4j logging. To use log4j anyway, set the com.stc.jms.logging system property to:

`com.stc.jms.util.LoggerFactoryApache`.

Chapter 4

Working with the Java API Samples

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

What's in This Chapter

4.1 About the Java Samples

The eGate API Kit provides Java code samples and Enterprise Designer Project samples designed to work together to demonstrate different types of JMS messaging using a Java 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 2. The table describes what each **.zip** file contains.

Table 2 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 sample Java CAPS Project that you can import into Enterprise Designer.

4.2 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 deliver messages between senders and receivers or between publishers and subscribers.

Before continuing, make sure you have downloaded the sample file as described in **"Installing the eGate API Kit" on page 9**. Implementing the sample Projects consists of the following steps:

4.2.1 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.

4.2.2 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*.

4.2.3 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*.

4.3 Building the Sample Java Applications

In order to compile the Java sample client application (or any client applications you create), you must edit the CLASSPATH variable to include the **.jar** files you downloaded during installation (see **"Post-Installation Instructions" on page 10**). Make sure this has been completed before performing the following steps.

You can modify the sample code before compiling it if desired.

To build the sample Java application

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).

   The samples are located in one of the following paths:

   - CodeSamples\apikit\Java (for Windows)
   - CodesamplesUNIX\apikit\Java (for UNIX)
3. From a command line, navigate to the directory where the code samples are located and run the following command:

   ```
   javac *.java
   ```

4.4 Running the Sample Java 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 3 lists each messaging type demonstrated in the samples along with their corresponding Java class names.

Table 3 Class Names for Java Samples

Messaging Type	Class Name
Queue Send	Sender
Queue Receive	Receiver
Queue Requestor	SampleQueueRequestor
Topic Publish	Publisher
Topic Subscribe	Subscriber
Topic Requestor	SampleTopicRequestor
Topic Selector Publish	SelectorPublisher
Topic Selector Subscribe	SelectorSubscriber
XA Publish	XAPublisher
XA Subscribe	XASubscriber

Each Java class requires the following flags:

- **-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)

The exceptions to this are the SampleQueueRequestor and SampleTopicRequestor classes, which require the host and port flags to be fully spelled out (**-host** and **-port**).

Important: *To run the Java applications, you must modify the CLASSPATH environment variable by adding the path to the class files you compiled.*

To run a send/receive or non-XA publish/subscribe sample application

1. Open two command line windows.
2. In the first command line, run the message consumer class; for example:

```
java Subscriber -h localhost -p 18007
```

3. In the second command line, run the message producer class; for example:

```
java Publisher -h localhost -p 18007
```

The text of the default message appears along with a comment that the message was processed.

To run an XA publish/subscribe sample application

1. Open two command line windows.
2. In the first command line, run the XA message consumer class; for example:

   ```
   java XASubscriber -h localhost -p 18007
   ```

3. In the second command line, run the XA message producer class; for example:

   ```
   java XAPublisher -h localhost -p 18007
   ```

4. In the message producer window, do one of the following:
   - To send the message, type **C** and press **Enter**.
   - To roll back the message, type **R** and press **Enter**.
5. If you committed the message, the subscriber window displays the text of the message along with a comment that it was processed. Do one of the following:
   - To receive the message, type **C** and press **Enter**.
   - To roll back the message, type **R** and press **Enter**.

Note: *If you do not commit the message, it remains in the queue to be reprocessed.*

6. After the message is received, do one of the following:
   - Send additional messages from the producer.
   - Close the consumer by typing **Q** and pressing **Enter**.

To run a requestor sample application

1. Open a command line window.
2. Run the requestor class; for example:

   ```
   java SampleQueueRequestor -host localhost -port 18007
   ```

   The text of the default message appears along with a comment that the message was processed.

Index

T

X