SYBASE®

Java Message Service User's Guide

EAServer

6.0

DOCUMENT ID: DC00486-01-0600-01

LAST REVISED: July 2006

Contents

About This Book

Subject

This book contains information about configuring the EAServer message service and Java Message Service (JMS) client applications.

Audience

This book is for anyone responsible for configuring the EAServer runtime environment, or for creating and deploying packages and components on EAServer.

How to use this book

Chapter 1, "Message Service Overview," describes the features of the EAServer message service.

Chapter 2, "Setting up the Message Service," explains how to use the Management Console to configure message queues, topics, listeners, connection factories, and listeners.

Chapter 3, "Developing JMS Clients," describes how to create JMS client applications.

Related documents

Core EAServer documentation The core EAServer documents are available in HTML and PDF format in your EAServer software installation and on the SyBooks™ CD.

What's New in EAServer 6.0 summarizes new functionality in this version.

The *EAServer API Reference Manual* contains reference pages for proprietary EAServer Java classes and C routines.

The *EAServer Automated Configuration Guide* explains how to use Ant-based configuration scripts to:

- Define and configure entities, such as EJB modules, Web applications, data sources, and servers
- Perform administrative and deployment tasks

The *EAServer CORBA Components Guide* explains how to:

- Create, deploy, and configure CORBA and PowerBuilder™ components and component-based applications
- Use the industry-standard CORBA and Java APIs supported by EAServer

The *EAServer Enterprise JavaBeans User's Guide* describes how to:

- Configure and deploy EJB modules
- Develop EJB clients, and create and configure EJB providers
- Create and configure applications clients
- Run the EJB tutorial

The *EAServer Feature Guide* explains application server concepts and architecture, such as supported component models, network protocols, server-managed transactions, and Web applications.

The *EAServer Java Message Service User's Guide* describes how to create Java Message Service (JMS) clients and components to send, publish, and receive JMS messages.

The *EAServer Migration Guide* contains information about migrating EAServer 5.*x* resources and entities to an EAServer 6.0 installation.

The *EAServer Performance and Tuning Guide* describes how to tune your server and application settings for best performance.

The *EAServer Security Administration and Programming Guide* explains how to:

- Understand the EAServer security architecture
- Configure role-based security for components and Web applications
- Configure SSL certificate-based security for client connections
- Implement custom security services for authentication, authorization, and role membership evaluation
- Implement secure HTTP and IIOP client applications
- Deploy client applications that connect through Internet proxies and firewalls

The *EAServer System Administration Guide* explains how to:

- Start the preconfigured server and manage it with the Sybase Management Console
- Create, configure, and start new application servers
- Define database types and data sources
- Create clusters of application servers to host load-balanced and highly available components and Web applications
- Monitor servers and application components

- Automate administration and monitoring tasks with command line tools

The *EAServer Web Application Programming Guide* explains how to create, deploy, and configure Web applications, Java servlets, and JavaServer Pages.

The *EAServer Web Services Toolkit User's Guide* describes Web services support in EAServer, including:

- Support for standard Web services protocols such as Simple Object Access Protocol (SOAP), Web Services Description Language (WSDL), and Uniform Description, Discovery, and Integration (UDDI)
- Administration tools for deployment and creation of new Web services, WSDL document creation, UDDI registration, and SOAP management

The *EAServer Troubleshooting Guide* describes procedures for troubleshooting problems that EAServer users may encounter. This document is available only online; see the EAServer Troubleshooting Guide at http://infocenter.sybase.com/help/index.jsp?topic=/com.sybase.help.eas_5.2.eastg/html/eastg/title.htm.

jConnect for JDBC documents EAServer includes the jConnect™ for JDBC™ 6.0.5 driver to allow JDBC access to Sybase database servers and gateways. The *jConnect for JDBC 6.0.5 Programmer's Reference* is available on the Sybase Product Manuals Web site at http://infocenter.sybase.com/help/index.jsp?topic=/com.sybase.help.jconnjdbc_6.05.prjdbc/html/prjdbc/title.htm&toc=/com.sybase.help.jconnjdbc_6.05/toc.xml.

Sybase Software Asset Management User's Guide EAServer includes the Sybase Software Asset Management license manager for managing and tracking your Sybase software license deployments. The *Sybase Software Asset Management User's Guide* is available on the Getting Started CD and in the EAServer 6.0 collection on the Sybase Product Manuals Web site at http://infocenter.sybase.com/help/index.jsp?topic=/com.sybase.help.eas_6.0/title.htm.

Conventions

The formatting conventions used in this manual are:

Formatting example	To indicate
commands and methods	When used in descriptive text, this font indicates keywords such as: • Command names used in descriptive text • C++ and Java method or class names used in descriptive text • Java package names used in descriptive text • Property names in the raw format, as when using Ant or jagtool to configure applications rather than the Management Console

Formatting example	To indicate
variable, *package*, or *component*	Italic font indicates: • Program variables, such as *myCounter* • Parts of input text that must be substituted, for example: `Server.log` (with *Server* italic) • File names • Names of components, EAServer packages, and other entities that are registered in the EAServer naming service
File | Save	Menu names and menu items are displayed in plain text. The vertical bar shows you how to navigate menu selections. For example, File | Save indicates "select Save from the File menu."
`package 1`	Monospace font indicates: • Information that you enter in the Management Console, a command line, or as program text • Example program fragments • Example output fragments

Other sources of information

Use the Sybase Getting Started CD, the SyBooks CD, and the Sybase Product Manuals Web site to learn more about your product:

- The Getting Started CD contains release bulletins and installation guides in PDF format, and may also contain other documents or updated information not included on the SyBooks CD. It is included with your software. To read or print documents on the Getting Started CD, you need Adobe Acrobat Reader, which you can download at no charge from the Adobe Web site using a link provided on the CD.
- The SyBooks CD contains product manuals and is included with your software. The Eclipse-based SyBooks browser allows you to access the manuals in an easy-to-use, HTML-based format.

  Some documentation may be provided in PDF format, which you can access through the PDF directory on the SyBooks CD. To read or print the PDF files, you need Adobe Acrobat Reader.

  Refer to the *SyBooks Installation Guide* on the Getting Started CD, or the *README.txt* file on the SyBooks CD for instructions on installing and starting SyBooks.

- The Sybase Product Manuals Web site is an online version of the SyBooks CD that you can access using a standard Web browser. In addition to product manuals, you will find links to EBFs/Maintenance, Technical Documents, Case Management, Solved Cases, newsgroups, and the Sybase Developer Network.

  To access the Sybase Product Manuals Web site, go to Product Manuals at http://sybooks.sybase.com/nav/base.do.

Sybase certifications on the Web

Technical documentation at the Sybase Web site is updated frequently.

❖ Finding the latest information on product certifications

1. Point your Web browser to Technical Documents at http://www.sybase.com/support/techdocs/.
2. Select Products from the navigation bar on the left.
3. Select a product name from the product list and click Go.
4. Select the Certification Report filter, specify a time frame, and click Go.
5. Click a Certification Report title to display the report.

❖ Creating a personalized view of the Sybase Web site (including support pages)

Set up a MySybase profile. MySybase is a free service that allows you to create a personalized view of Sybase Web pages.

1. Point your Web browser to Technical Documents at http://www.sybase.com/support/techdocs/.
2. Click MySybase and create a MySybase profile.

Sybase EBFs and software maintenance

❖ Finding the latest information on EBFs and software maintenance

1. Point your Web browser to the Sybase Support Page at http://www.sybase.com/support.
2. Select EBFs/Maintenance. If prompted, enter your MySybase user name and password.
3. Select a product.
4. Specify a time frame and click Go. A list of EBF/Maintenance releases is displayed.

Padlock icons indicate that you do not have download authorization for certain EBF/Maintenance releases because you are not registered as a Technical Support Contact. If you have not registered, but have valid information provided by your Sybase representative or through your support contract, click Edit Roles to add the “Technical Support Contact” role to your MySybase profile.

5 Click the Info icon to display the EBF/Maintenance report, or click the product description to download the software.

Accessibility features

EAServer has been tested for compliance with U.S. government Section 508 Accessibility requirements. The online help for this product is also provided in Eclipse help formats, which you can navigate using a screen reader.

The Web console supports working without a mouse. For more information, see “Keyboard navigation” in Chapter 2, “Management Console Overview,” in the *EAServer System Administration Guide*.

The Web Services Toolkit plug-in for Eclipse supports accessibility features for those that cannot use a mouse, are visually impaired, or have other special needs. For information about these features see the Eclipse help:

1 Start Eclipse.

2 Select Help | Help Contents.

3 Enter `Accessibility` in the Search dialog box.

4 Select Accessible User Interfaces or Accessibility Features for Eclipse.

Note You may need to configure your accessibility tool for optimal use. Some screen readers pronounce text based on its case; for example, they pronounce ALL UPPERCASE TEXT as initials, and MixedCase Text as words. You might find it helpful to configure your tool to announce syntax conventions. Consult the documentation for your tool.

For additional information about how Sybase supports accessibility, see Sybase Accessibility at http://www.sybase.com/accessibility. The Sybase Accessibility site includes links to information on Section 508 and W3C standards.

CHAPTER 1

Message Service Overview

The EAServer message service supports version 1.1 of the Java Messaging Service (JMS) specification.

For more details on JMS, see the JMS specifications from Sun Microsystems at http://java.sun.com/products/jms/.

Introduction

The message service allows you to send or publish messages to a queue or topic, where they are stored until they can be delivered to either a client or a component. The message service supports two messaging models, point-to-point and publish/subscribe. Point-to-point messaging sends a message to one consumer. The publish/subscribe model publishes messages that are available to all consumers who subscribe to the message topic. You can receive messages using message listeners. The message service provides transient and persistent message storage for queued messages.

A messaging service provides a flexible solution for many business needs. As a practical example, consider a scenario in which a supplier must communicate prices to a number of retailers, and the retailers want to place orders for items at updated prices.

The message service requires a listener to respond to incoming orders, and a way to manage the list of retailers and add new ones. You must ensure that each party receives and processes all the transactions that are sent its way. If one of the retailers is offline, or network routing to its server fails, your application must continue trying to establish communications until the transaction can be completed. You must provide persistence for critical transactions until all recipients acknowledge them. You also want to ensure that all parties are granted access, are who they say they are, and that transactions are secure during transmission.

Many business transactions take place in an environment such as this, where connectivity cannot be guaranteed and transactions require security. Inserting a messaging service between business nodes in a network insulates your application code from these issues.

The key features of the EAServer message service include:

- High availability and load balancing
- Message security
- Reliable delivery
- Scalable notification
- Transaction management

High availability and load balancing

The message service uses server clustering to provide high availability and load balancing. For complete information about server clustering, see Chapter 6, "Clusters and Synchronization," in the *EAServer System Administration Guide*.

Message security

The message service provides role-based security for message queues and message topics. The message service operations and the access categories required to use them are:

Access category	Message service operations
Consumer	Receiving messages and creating topic subscriptions
Producer	Sending and publishing messages

You can assign permissions for access categories separately. See "Message queues and topics" on page 8.

Reliable delivery

To ensure reliable message delivery, the message service provides:

- IIOP/IIOPS connections for client notification.
- HTTP tunneling of IIOP connections. Messages can be delivered through client-side firewalls that accept only HTTP/HTTPS.
- Persistent messages that guarantee message delivery, subject to the reliability of the persistent store.

Scalable notification

Connection factory properties are available to support batching of both producer (send and publish) and consumer (receive and acknowledge) operations.

Transaction management

A transactional operation runs in the caller's transaction, or, if the caller is not enlisted in a transaction, in a new transaction. The following operations can be transactional:

- **publish** A message producer can publish a message within a transaction.
- **send** A message producer can send a message within a transaction.
- **acknowledge** A client can acknowledge and process a message in the same transaction.
- **onMessage** A listener component can process a message within the same transaction as the automatic acknowledgment, which occurs when onMessage returns.
- **move** A message can be moved from one queue to another only within a transaction.

Client applications

To create EAServer message service client applications, you can use either:

- JMS 1.1 API – see Chapter 3, "Developing JMS Clients," or
- CtsComponents::MessageService CORBA API – see the HTML interface documentation in the *html/ir* subdirectory of your EAServer installation.

New features

New messaging features in EAServer 6.0 include:

- Support for the complete JMS 1.1 API.
- Message-driven beans (MDBs) can receive message types other than JMS. A J2EE Connector Architecture (JCA) 1.5 resource adapter integrates messaging providers with the container.
- Messages can be processed in the order in which they are received, even when multiple consumers access the same queue.
- Duplicate messages can be ignored.
- Application servers running on a mobile device can initiate pull and push operations; both online and offline modes are supported.
- Header and system properties can be used to construct source or target queue addresses; for pull, only system properties can be used.
- You can log in to the JMS provider using IIOP single sign-on scripts.
- For JMS clients, a new initial context factory class com.sybase.jms.client.InitialContextFactory replaces com.sybase.jms.InitialContextFactory.
- The com.sybase.jms.client.InitialContextFactory class supports four provider URL forms:
  - `iiop://`***`host:port`*** – uses IIOP to access the message service.
  - `jms-provider:`***`jmsProviderName`*** – uses the named JMS provider.
  - `run-server.`***`server_name`*** – uses IIOP to access the message service, starts the server in-process if it is not already running, and selects the server's first IIOP listener as the provider URL.
  - `start-server.`***`server_name`*** – uses IIOP to access the message service, starts the server out-of-process if it is not already running, and selects the server's first IIOP listener as the provider URL.
- The com.sybase.jms.client.InitialContextFactory class performs JNDI lookups to resolve names that are not preconfigured:
  - A name ending with "queue" is assumed to be the name of a JMS message queue.
  - A name ending with "topic" is assumed to be the name of a JMS message topic.

- A name ending with “queueconnectionfactory” is assumed to be the name of a JMS queue connection factory.
- A name ending with “topicconnectionfactory” is assumed to be the name of a JMS topic connection factory.
- Any other name ending with “connectionfactory” is assumed to be the name of a common JMS connection factory.

Note Name comparisons are case-insensitive.

CHAPTER 2 Setting up the Message Service

Before you can use the message service to send and receive messages, you must add and configure the message service parts. Once you configure the message service, it is available to every server that you create.

Topic	Page
Message queues and topics	8
Connection factories	16
Listeners	19
Message selectors	23
Threads	23

You can use the Management Console to add and configure message service parts:

- Permanent destinations – message queues and topics. Add one message queue for each message recipient. To identify the subject of messages to which you want to subscribe, add message topics. You can also create message queues and topics using the CORBA API or the JMS createQueue and createTopic methods. Although these JMS methods are not portable, creating message queues and topics programmatically can significantly reduce the system administrator's work.

  To restrict access to a message queue, define security roles for message queue producers and consumers. You can also add security roles to a topic.

- Connection factories – connection factories enable JMS client applications to establish connections with the message service.
- Message listeners – to provide asynchronous message notification for clients and components, implement a message listener, and install it on a message queue or topic.

Message queues and topics

To provide permanent destinations for JMS client applications, use the Management Console to define message queues and message topics. When you create a message queue or topic, you can optimize its configuration properties, which benefits every JMS client application that uses the destination.

Message queues

The EAServer JMS provider allows you to create and configure JMS message queues; however, you need not preconfigure message queues unless their properties require nondefault values. A JMS client that uses the JMS initial context factory com.sybase.jms.client.InitialContectFactory can look up preconfigured queues by name. To look up queues that have not been preconfigured, clients can either use a JNDI name that ends with “Queue,” or use the javax.jms.Session.createQueue API. You can also use this API to look up preconfigured queues.

To override default properties for JMS temporary queues, name the message queue “javax.jms.TemporaryQueue.”

❖ Adding a message queue

1. In the Management Console, expand the Resources folder, select JMS Message Queues, right-click, and select Add.
2. In the wizard, enter a name for the message queue, and select Finish.

❖ Configuring message queue properties

1. Select the message queue to configure.
2. In the right pane, configure the properties on these tabs:
   - General tab
   - Security tab

General tab

Table 2-1 defines the properties you can set on the General tab for message queues and topics.

Table 2-1: Message queue and topic general properties

Property	Datatype	Default value	Description
Queue ID	string	none	Message queues only. A short name for the message queue.
Cluster Partition	string	default	The name of the cluster partition that serves the message queue. The "default" cluster partition contains all the servers in the cluster. If a server fails, another server in the same partition takes over management of the queue. If the server is not in a cluster, this property is ignored.
Data Source	string	message.db	The name of a data source, which identifies the database where persistent and transient overflow messages are stored. If a message queue uses a different data source than other resources involved in a transaction (such as direct JDBC access or EJB entity beans), all the data sources must be configured for two-phase commit to achieve atomicity between message queue operations and other resource access. Sybase recommends that you not change this property value.
Database Table	string	jms_pm	The name of the database table in which persistent and transient overflow messages for this message queue are stored.
Automatic Recovery Timeout	long	60	The number of seconds before unacknowledged messages are recovered. When a client receives a message from a queue, but does not acknowledge it, the message is recovered automatically so that it can be received again. If the timeout is exceeded, the client is assumed to have failed. If the messages for a particular queue require a long time for clients to process, increase this value accordingly.
Duplicate Detection Key	string	JMSMessageID	To detect duplicate messages, the message service compares the value in this database field.
Duplicate Detection Protocol	string	optimistic	Select the protocol to use for detecting duplicate messages: optimistic, pessimistic, or none. To optimize the duplicate detection by using deferred inserts, and checking for duplicate key exceptions (in the duplicate detection table), set to optimistic. If duplicates are detected, transactions are rolled back, and automatic transaction retry adds non-deferred duplicate checking. To always use non-deferred duplicate checking, set to pessimistic. When there are few duplicates, optimistic is more efficient. When there are many duplicates, pessimistic may be more efficient.
Duplicate Detection Table	string	jms_dd	The database table in which the Duplicate Detection Key exists.

Property	Datatype	Default value	Description
Duplicate Detection Timeout	long	3600 seconds	The number of seconds before an attempt to detect a duplicate message times out.
Idle Connection Timeout	long	60	The number of seconds a queue remains in memory when it is idle; that is, when it is not being accessed by a client or a server component. Set to 0 or a negative number for no timeout. Transient messages that are in memory when a timeout occurs are discarded.
Maximum Messages In Memory	long	10	The maximum number of messages to store in memory for this message queue. Typically, keeping messages in memory improves performance. Persistent messages are always stored in the database; some are additionally held in memory. Nonpersistent (transient) messages are held in memory, and stored in the database only if both the value of maximumMessagesInMemory is exceeded and the value of storeTransientMessages is true. Nonpersistent messages are discarded from memory if both the value of maximumMessagesInMemory is exceeded and the value of storeTransientMessages is false.
Maximum Received Messages Waiting For Acknowledgment	long	0	The maximum number of messages that can be received from this message queue without being acknowledged. Once the maximum is reached, no more messages can be received by clients until the outstanding messages have been acknowledged. If this property (maximumWaitingForAcknowledge) value is either set to 0, or greater than the value of maximumMessagesInMemory, the value of maximumMessagesInMemory takes precedence.

Property	Datatype	Default value	Description
Pull Messages From	string	none	To pull messages into a message queue from a queue on a remote JMS provider, or to pull messages into a topic from a topic on a remote JMS provider, specify the queue or topic name and the provider using: *remote-destination@provider-name* where *remote-destination* identifies one or more remote destinations (queues or topics) and *provider-name* is the name of a configured JMS provider. Specify each remote destination as either: • `queue:`***`queue-name`*** • `topic:`***`topic-name`*** Specify multiple destinations in a comma-separated list. Persistent messages are sent exactly once if you are either using a JMS provider configured with a com.sybase.jms.client.InitialContextFactory type Initial Context Factory, or if the JMS provider uses a resource adapter that supports two-phase commit. Otherwise, messages are sent at least once. The remote message queue or topic name can contain one or more substitution expressions of the form "${*var*}," where *var* is the name of a local Java system property.

Property	**Datatype**	**Default value**	**Description**
Push Messages To	string	none	To push messages from this queue to one or more queues or topics on a remote JMS provider, specify the provider and destination using: *remote-destination@provider-name* where *remote-destination* identifies one or more remote destinations (queues or topics) and *provider-name* is the name of a configured JMS provider. Specify each remote destination as either: • `queue:`***`queue-name`*** • `topic:`***`topic-name`*** Specify multiple destinations in a comma-separated list. Persistent messages are sent exactly once if you are either using a JMS provider configured with a com.sybase.jms.client.InitialContextFactory type Initial Context Factory, or if the JMS provider uses a resource adapter that supports two-phase commit. Otherwise, messages are sent at least once. A destination name can contain one or more substitution expressions of the form "${*var*}," where *var* is the name of either a local Java system property or a JMS message header property. If a property referenced by a substitution expression exists as both a message header property and a system property, the message header property takes precedence. **Note** Using message header properties to identify the destination queue or topic provides a limited form of content-based routing.
Compress Stored Messages	boolean	false	Select to compress messages before saving in the database. Enabling message compression increases CPU usage of the application server, and reduces the CPU and disk usage of the database server. Sybase recommends that you test system performance to determine whether enabling message compression is beneficial.
Ignore Duplicate Messages	boolean	false	This property is currently not used.

Property	Datatype	Default value	Description
Process Messages In Order	boolean	false	Indicates whether to process messages strictly in the order that they are received. Messages are usually processed in order; however, when multiple receivers process the same queue, or messages are received and not acknowledged (then subsequently recovered), there is a possibility of processing messages out of order. Setting this value to true is equivalent to setting Maximum Received Messages to 1, which ensures that messages are processed in sequence, one at a time.
Store Transient Messages	boolean	false	Select to store transient (nonpersistent) messages in the message store when either the queue overflows—see Maximum Messages In Memory—or the queue times out due to client inactivity. If not selected and the queue either overflows or times out, messages are discarded.
Suspend Message Delivery	boolean	false	Select to suspend message delivery for this queue.

Security tab

Select the Security tab, and configure the properties defined in Table 2-2.

Table 2-2: Message queue and topic security properties

Property	Datatype	Default value	Description
Producer Roles	string	none	To send messages to a message queue, or to publish messages with the specified topic, a client must be a member of at least one of the security roles. • Select – select individual security roles from the list. • All – clients must be a member of at least one of the listed security roles. • None – anyone can send messages to the queue, or publish messages with the topic.
Consumer Roles	string	none	To browse or receive messages from this queue, or to receive messages with the specified topic, a client must be a member of at least one of the security roles. • Select – select individual security roles from the list. • All – clients must be a member of at least one of the listed security roles. • None – anyone can read messages in this queue, or receive messages with the specified topic.

❖ **Deleting a message queue**

1. Remove any listeners attached to the queue.
2. Expand the JMS Message Queues folder, then select the message queue to delete.
3. Right-click the queue, and select Delete.
4. In the wizard, confirm that you want to delete the queue.

Message topics

The EAServer JMS provider allows you to create and configure JMS message topics. You need not preconfigure message topics unless their properties require nondefault values. A JMS client that uses the JMS initial context factory com.sybase.jms.client.InitialContectFactory can look up preconfigured topics by name. To look up topics that have not been preconfigured, clients can either use a JNDI name that ends with “Topic,” or use the javax.jms.Session.createTopic API. You can also use this API to look up preconfigured topics.

Each message topic subscription is managed internally by a corresponding message queue, which is called a **subscription queue**. The property values for each subscription queue are the same as the property values for the corresponding topic. In other words, topic properties are used as templates for the properties of the internal subscription queues. Internal subscription queues have names that begin with either “ds~” (for durable subscriptions) or “ts~” (for transient subscriptions). You may see these names when viewing statistics or browsing the jms_pm or jms_ts database tables. Sybase recommends that you do not explicitly configure properties for internal subscription queues—instead, set topic properties to indirectly configure the internal subscription queues.

❖ **Adding a message topic**

To override default properties for JMS temporary topics, name the topic “javax.jms.TemporaryTopic.”

1. Expand the Resources folder, select JMS Message Topics, right-click, and select Add.
2. In the wizard, enter a name for the message topic, and select Finish.

❖ **Configuring message topic properties**

1. Select the message topic to configure.

2 In the right pane, select the General tab, and enter the topic properties described in Table 2-1 on page 9. The property values apply to both the message topic and the corresponding subscription queue.

3 Select the Security tab, and configure the properties defined in Table 2-2 on page 13.

❖ **Deleting a message topic**

1 In the JMS Message Topics folder, select the topic to delete, right-click, and select Delete.

2 In the wizard, confirm that you want to delete the message topic.

Viewing message statistics

You can view message statistics using either a spreadsheet program or the Management Console.

❖ **Viewing message statistics using a spreadsheet program**

- In a spreadsheet program, use a Web query to import statistics. For example, in Microsoft Excel, select Data | Import External Data | New Web Query. As the URL, enter `http://`***`host`***`:`***`port`***`/wsh/run?command=get-statistics`, where *host* and *port* are the server's host name and HTTP port, respectively.

❖ **Viewing message queue and topic statistics using the Management Console**

1 Expand the Servers folder, select the server, then select Statistics.

2 To view message queue statistics, select the MessageQueue tab—see "MessageQueue tab" in Chapter 11, "Runtime Monitoring," in the *EAServer System Administration Guide*.

To view message topic statistics, select the MessageTopic tab—see "MessageTopic tab" in Chapter 11, "Runtime Monitoring," in the *EAServer System Administration Guide*.

Connection factories

To enable JMS applications to establish connections with the message service, you can create queue connection factories for point-to-point messaging, topic connection factories for publish/subscribe messaging, and generic connection factories, which you can use to create connections for both types of messaging.

A JMS client application can also use a connection factory that has not been preconfigured—see “Looking up a connection factory” on page 28.

You can use the Management Console to add and configure connection factories.

❖ Adding a JMS connection factory

1. In the Management Console, expand the Resources folder.
2. To add a generic connection factory, select JMS Connection Factories; to add a queue connection factory, select JMS Queue Connection Factories; to add a topic connection factory, select JMS Topic Connection Factories.
3. Right-click, and select Add.
4. Enter a name for the connection factory, and select Finish.

❖ Configuring JMS connection factories

1. In the Management Console, expand the Resources folder.
2. Select one of JMS Connection Factories, JMS Queue Connection Factories, or JMS Topic Connection Factories.
3. Highlight the connection factory you want to configure.
4. On the General tab, enter the connection factory properties described in Table 2-3.

Table 2-3: Connection factory properties

Property	Datatype	Default value	Description
Client ID	string	blank	Optional. If a JMS client uses a connection factory with an unspecified clientID, the client use its TCP/IP host name as the clientID. If this is insufficient, either: • The client can call the javax.jms.Connection.setClientID method to set the clientID, or • The administrator can create a distinct connection factory for each client. Sybase does not recommended this method, as it can make the administrator unnecessarily busy.

Property	Datatype	Default value	Description
Command Batch Size	long	0	Some communication between a JMS client and the JMS server can be batched to improve message throughput (possibly at the expense of increasing message delivery times). The commandBatchSize property allows you to configure the maximum number of commands in a batch. A value of 0 indicates that the client should supply its own value; for example, 20. Sybase recommends that you test different values to determine which value provides the best performance in your environment. Different client applications might benefit from different values. Values that are too large may cause a client to consume excessive amounts of memory. A reasonable starting value is 20.
Command Batch Wait	long	1	The maximum time (in milliseconds) that the client waits for a batch to fill up before sending it to the server, partially filled. The configuration property name is commandBatchWait. Sybase recommends that you test different values, to determine which value provides the best performance in your environment. Values that are too large may cause delays in message delivery. A reasonable starting value is 1 millisecond.
Receive Batch Size	long	0	A JMS client runtime may receive messages in batches to improve communication with the JMS server. If set to 0, clients must provide their own value. The configuration property name is receiveBatchSize. Sybase recommends that you test different values to determine which value provides the best performance in your environment. Values that are too large may cause a client to consume excessive amounts of memory. A reasonable starting value is 20. **Note** If a client receives only a small number of messages, then closes its connection, a value of 20 may be too large, because prefetched messages would need to be “unreceived.” In such cases, set the value to 1, then increase in small increments until you determine the best performance value.

Property	Datatype	Default value	Description
Temporary Queue Template	string	javax.jms.TemporaryQueue	If a session associated with this connection factory creates a JMS temporary queue, the queue's configuration properties are copied from this named queue. The configuration property name is temporaryQueueTemplate.
Temporary Topic Template	string	javax.jms.TemporaryTopic	If a session associated with this connection factory creates a JMS temporary topic, the topic's configuration properties are copied from this named topic. The configuration property name is temporaryTopicTemplate.
Batch Persistent Messages	boolean	false	Select to enable client-side batching of persistent messages. This may improve throughput for applications that use **transacted sessions**, but should be used only if the client can resend (potentially) lost messages upon reconnection. The configuration property name is batchPersistentMessages. Within a transaction, if all a client's sent messages are derived from messages that were received in the same transaction, you can safely enable this option, even though client failure might result in batched transaction commits not being sent to the server. This is because when a client restarts, messages that were previously received but not acknowledged are received again, and any lost transactions can be reissued.
Batch Transaction Commits	boolean	false	Select to enable client-side batching of transaction commits. This may improve throughput for applications that use transacted sessions, but should be used only if the client can resend (potentially) lost messages upon reconnection. The configuration property name is batchTransactionCommits. Within a single transaction, if all of a client's sent messages are derived from messages received in the same transaction, then this option can be safely enabled, even though client failure can result in batched-transaction commits not being sent to the server. This is because upon client restart, messages that were previously received but not acknowledged are received again, and any lost transactions can be reissued.

Property	Datatype	Default value	Description
Disable Transient Exceptions	boolean	false	To optimize the performance of a JMS application, set to false. The configuration property name is disableTransientExceptions. Transient exceptions occur when the JMS client runtime determines that the failure or restart of a server may have resulted in the loss of some nonpersistent messages. If transient exceptions occur, the client can handle them, or you can install an exception listener (using the javax.jms.Connection.setExceptionListener method) to handle them. If clients do not care about losing nonpersistent messages, set to true.

Listeners

Message listeners allow you to receive messages asynchronously. Once you have implemented a listener and installed it on a message consumer, the listener can send the message to other consumers, or notify one or more components. A message listener can be:

- A client-side JMS message listener that is associated with a consumer using javax.jms.MessageConsumer.setMessageListener. See "Implementing and installing message listeners" on page 40.
- An EJB message-driven bean (MDB) that implements the javax.jms.MessageListener interface. See "Message-driven beans" on page 19.
- An EAServer class that implements the CtsComponents::MessageListener interface. See the HTML documentation in the *html/ir* subdirectory of your EAServer installation.

Message-driven beans

An MDB is a type of Enterprise JavaBean (EJB) specifically designed as a message consumer. You can install an MDB as a listener on a message queue or topic. MDBs can listen for messages from either the EAServer JMS provider or a JCA 1.5 resource adapter.

By listening for messages from a resource adapter, MDBs can receive message types other than JMS. Resource adapters also enable an MDB to listen for messages from third-party JMS providers.

MDBs implement the javax.jms.MessageListener interface, which contains only the onMessage method. This example illustrates the skeleton code for a message listener:

```
class QueueMessageListener implements MessageListener
{
   public void onMessage(javax.jms.Message msg)
   {
      // process message, notify component
   }
}
```

Unlike other EJBs, message-driven beans do not have a home or remote interface, and clients cannot directly access an MDB. When an MDB is installed as a listener on a message queue or topic and a message arrives, EAServer instantiates the MDB and calls onMessage to notify the MDB that a message has been delivered to the queue or topic on which it is installed.

MDB interface methods

An MDB must implement the MessageDrivenBean interface, which consists of the following methods:

Method name	**Description**
ejbCreate	Creates an instance of an MDB.
setMessageDrivenContext	Associates an MDB instance with its context, which EAServer maintains. This provides the MDB instance access to the MessageDrivenContext interface.
ejbRemove	Notifies the MDB instance that it is being removed and should release its resources.

An MDB instance with container-managed transactions can call these MessageDrivenContext interface methods:

Method name	**Description**
setRollbackOnly	To specify that the current transaction must be rolled back.
getRollbackOnly	To test whether the current transaction has been marked to roll back.
getUserTransaction	Returns the javax.transaction.UserTransaction interface, with which the MDB can set and obtain transaction status.

For information about Enterprise JavaBeans, see the *EJB User's Guide*.

❖ Installing and configuring an MDB as a message listener

1 Deploy the EJB-JAR file that contains the MDB, as described in Chapter 2, “Deploying and Configuring EJB Components,” in the *EJB User’s Guide.*

   When you deploy an EJB-JAR, EAServer creates an Ant configuration file, which contains the component property settings read from the EJB-JAR deployment descriptor. You may need to customize these settings.

2 The following example sets properties for the MDB called “MyQueueListener,” which is implemented by the ejb.components.cmstests.MyQueueListener component:

```
<setProperties component="ejb.components.cmstests.MyQueueListener">
  <property name="ejbName" value="MyQueueListener" />
  <property name="ejbClass"
     value="ejb.components.cmstests.MyQueueListener_EJB" />
  <property name="j2eeType" value="MessageDrivenBean" />
  <messageListener queue="MyQueueListenerQueue" name="" />
  <activationConfig name="acknowledgeMode" value="Auto-acknowledge" />
  <classLoader name="ejb.components.cmstests" />
  <threadMonitor name="${ejb.serviceThreadMonitor}" />
  <instancePool timeout="${ejb.poolTimeout}" />
  <transaction type="NotSupported" batch="${ejb.transactionBatch}"
     retry="${ejb.transactionRetry}" />
</setProperties>
```

   To assign the MDB as a listener for a message queue or topic:

   a In the Management Console, expand the EJB Modules folder, and select the package that contains the MDB.

   b On the User Configuration tab, find the messageListener definition inside the <setProperties> tags for the MDB component.

      To assign the listener to a message queue, set the value of messageListener queue to the name of a message queue; for example:

```
<setProperties component="ejb.components.cmstests.MyQueueListener">
  <messageListener queue="cts-mdb-test-queue" topic=""
     threadCount="1" />
</setProperties>
```

      To assign the listener to a message topic, set the value of messageListener topic to the name of a message topic; for example:

```
<setProperties component="ejb.components.cmstests.MyQueueListener">
  <messageListener queue="" topic="cts-mdb-test-topic"
```

```
        threadCount="1" />
    </setProperties>
```

3 To configure the MDB to listen on the inbound resource adapter of a JCA 1.5 connector, instead of the JMS provider, set the value of messageListener name to the name of a com.sybase.djc.connector.MessageListener component; for example:

```
<setProperties component="ejb.components.cmstests.MyQueueListener">
  <messageListener name="myConnectorMessageListener" />
</setProperties>
```

4 Reconfigure or recompile the MDB properties—see "Updating component properties" in Chapter 2, "Deploying and Configuring EJB Components," in the *EAServer Enterprise JavaBeans User's Guide*.

Managing dead messages

A message is considered "dead" when repeated attempts to receive the message fail and the transactions roll back.

You can specify transaction retry parameters in the Ant configuration script that defines the message listener; for example:

```
<setProperties component="ejb.components.mymodule.MyListener">
    ...
    <transaction retry="true" retryCount="5" retryDelay="60"/>
 </setProperties>
```

Dead messages for a message queue named *xxx* are moved to a message queue named deadMessages:*xxx*. To view the messages in a deadMessages:*xxx* queue, you can use the JMS API QueueBrowser (see "Browsing messages" on page 41). To move a message from deadMessages:*xxx* to another message queue:

1 Obtain a transacted session (see "Creating sessions" on page 31).

2 Receive the message from deadMessages:*xxx* (see "Receiving messages" on page 40).

3 Acknowledge the message.

4 If the message should be recovered, send it to the *xxx* message queue; otherwise, send it to any message queue (see "Sending messages" on page 38).

5 Commit the transacted session.

You can download the JMS 1.1 API Specification from the JMS Download site at http://java.sun.com/products/jms/docs.html.

To check the message store for dead (persistent) messages, you can run an SQL query, such as:

```
select pm_qid, count(*) from jms_pm
where pm_qid like '%'
group by pm_qid
```

Message selectors

To filter the messages you receive and to subscribe to specific message topics, use a message selector.

Message selectors must conform to the JMS selector specification, which is a subset of the SQL-92 syntax—see the JMS 1.1 Specification at http://java.sun.com/products/jms/.

❖ **Adding a message selector to an MDB**

- Edit the user configuration script for the EJB-JAR in which the MDB component is defined.

  For example, to receive all published messages with the stock symbol “SY,” use the following message listener configuration:

```
<setProperties component="ejb.components.cmstests.MyTopicListener">
   <messageListener queue="" topic="StockPrice"
     selector="symbol = 'SY'" durable="true" threadCount="1" />
</setProperties>
```

Threads

You can define the number of threads dedicated to an MDB listener.

❖ **Defining the number of listener threads**

- In the user configuration script for the EJB-JAR in which the MDB is defined, set the value of the threadCount property; for example:

```
<setProperties component="ejb.components.cmstests.MyTopicListener">
```

```
    <messageListener threadCount="1" />
</setProperties>
```

Note To enable EAServer to create multiple instances of the MDB, set the threadCount property to a value greater than 1.

CHAPTER 3 Developing JMS Clients

Client runtime requirements

To run JMS clients, you must have:

- An EAServer client runtime installation
- A Java Runtime Environment (JRE) or JDK installation, version 1.4.2 or later

JMS client program flow

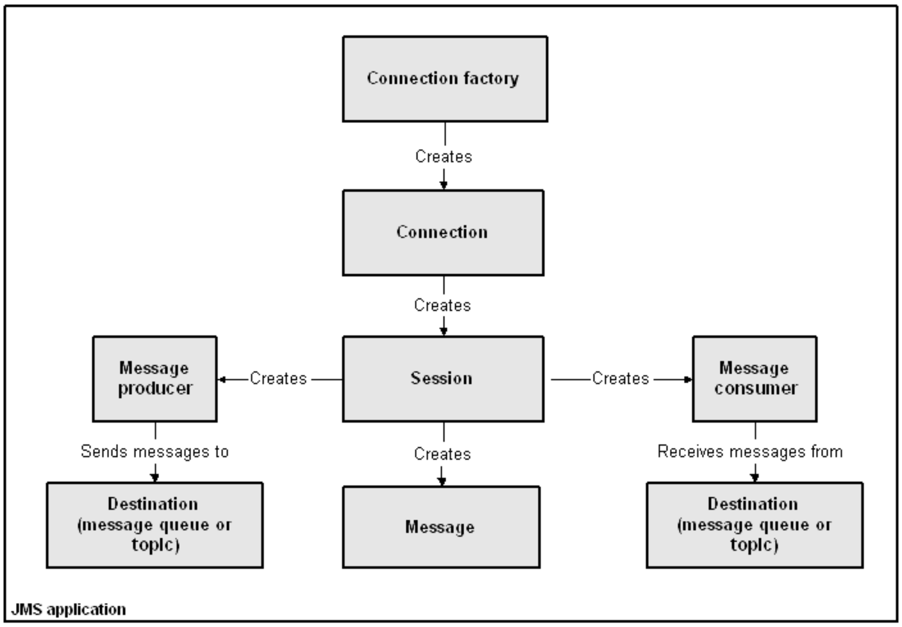
The steps in the table below describe the program flow in a typical JMS client:

Step	Action	For more information
1	Define the initial naming context.	See “Defining the initial naming context” on page 26.
2	Obtain a connection factory.	See “Looking up a connection factory” on page 28.

Step	Action	For more information
3	Create a connection.	See “Creating connections” on page 29.
4	Create one or more sessions.	See “Creating sessions” on page 31.
5	Look up message queues and topics.	See “Looking up destinations” on page 33.
6	Create message producers and consumers.	See “Creating message producers and consumers” on page 33.
7	Send and receive messages.	See “Sending and receiving messages” on page 37.

Figure 3-1 illustrates the relationship of the objects in a JMS application.

Figure 3-1: JMS application



Defining the initial naming context

A JMS client application must instantiate a Sybase InitialContext, and provide information to connect to a JMS provider.

The core JNDI interface used by client applications is javax.naming.Context, which represents the initial naming context used to resolve names to connection factories, message queues, and topics. To obtain an initial naming context:

1. Initialize a java.util.Properties instance:

```
java.util.Properties props = new java.util.Properties()
```

2. Set the InitialContext.INITIAL_CONTEXT_FACTORY property:

```
props.put(Context.INITIAL_CONTEXT_FACTORY,
      "com.sybase.jms.client.InitialContextFactory")
```

3. To define the connection properties, either specify a JMS provider or set the properties manually. EAServer includes a JMS provider named "default."

   To use the connection properties defined by the default JMS provider:

   a. Initialize an instance of a JMS provider:

```
com.sybase.jms.client.JmsProvider _jmsProvider =
      new com.sybase.jms.client.JmsProvider()
```

   b. Set the InitialContext.PROVIDER_URL property to "jms-provider:default":

```
props.put(Context.PROVIDER_URL, "jms-provider:default")
```

   This provides the information that is needed to connect to the default JMS provider. You can configure the properties of the default JMS provider and create new providers using the Management Console—see "Creating JMS providers" on page 42.

   To set the connection properties manually, set the URL for the server's IIOP port, the user name (principal), and the password (credentials):

```
props.put(Context.PROVIDER_URL, "iiop://myhost:2000")
props.put(Context.SECURITY_PRINCIPAL, "jmsuser")
props.put(Context.SECURITY_CREDENTIALS, "jmspass1")
```

4. Create the InitialContext instance:

```
return new InitialContext(props)
```

The JMS provider uses properties defined in the local client installation, not the server installation. You cannot use a JMS provider in an applet, since this feature requires access to configuration files in the EAServer installation. To run a JMS client application that uses a JMS provider, verify that the operating system library search path includes the server's *lib* subdirectory. For example, on Windows, PATH must include *%DJC_HOME%\lib*, and on UNIX, LD_LIBRARY_PATH must include *$DJC_HOME/lib*.

If you are creating a client application that must be portable to other servers, use an external mechanism to specify properties, rather than hard-coding values in the source code. For example, in a Java application, use command line arguments or a serialized Java properties file. To specify properties used by a Java applet, use parameters in the HTML Applet tag that loads the applet.

This example runs the JMS client application JMSClientClass and sets the InitialContext factory, URL, user name, and password properties at runtime:

```
java
-Djava.naming.factory.initial=com.sybase.jms.client.InitialContextFactory
-Djava.naming.provider.url=iiop://myhost:9000
-Djava.naming.security.principal="jmsuser"
-Djava.naming.security.credentials="jmspass1"
JMSClientClass
```

Looking up a connection factory

A connection factory allows you to create connections with the EAServer JMS provider, and specify a set of configuration properties that define the connections.

Beginning in version 6.0, EAServer provides a common connection factory that you can use to create both queue and topic connections. Queue connections allow you to send and receive messages using the point-to-point messaging model. Topic connections allow you to publish and receive messages using the publish/subscribe messaging model.

To create and configure connection factories, use the Management Console—see "Connection factories" on page 16. However, you need not preconfigure connection factories unless their properties require nondefault values.

To look up preconfigured connection factories by name, use the JMS initial context factory com.sybase.jms.client.InitialContectFactory.

To look up connection factories that have not been preconfigured:

- Common connection factories – use a JNDI name that ends with “ConnectionFactory”; the name cannot end with either “QueueConnectionFactory” or “TopicConnectionFactory.”
- Queue connection factories – use a JNDI name that ends with “QueueConnectionFactory.”
- Topic connection factories – use a JNDI name that ends with “TopicConnectionFactory.”

The following example illustrates how clients can use JNDI to look up a connection factory object, where *ctx* is the initial naming context:

```
// Look up a common connection factory that has not been preconfigured.
// You can use this connection factory to create queue and topic connections.

ConnectionFactory myConnFactry=
 (ConnectionFactory) ctx.lookup("MyConnectionFactory");

// Look up a preconfigured queue connection factory

QueueConnectionFactory queueCF =
 (QueueConnectionFactory) ctx.lookup("MyTestQueueCF");

// Look up a preconfigured topic connection factory

TopicConnectionFactory topicCF =
 (TopicConnectionFactory) ctx.lookup("MyTestTopicCF");
```

If the connection factory cannot be found, EAServer throws a javax.naming.NamingException.

Creating connections

Beginning in version 6.0, EAServer provides a common connection type that you can use for both message queues and topics.

To create a connection to the EAServer JMS provider, a JMS client must have access to a ConnectionFactory object. See “Looking up a connection factory” on page 28.

Once you have created a connection, you must explicitly start it before EAServer can deliver messages on the connection. To avoid message delivery before a client has finished setting up, you may want to delay starting the connection. This code fragment uses a common connection factory to create a connection, then starts the connection:

```
Connection myConnect = myConnFactry.createConnection();

// other setup procedures

myConnect.start();
```

You can stop message delivery using the Connection.stop method, then use start to resume delivery. While a connection is stopped, receive calls do not return with a message, and messages are not delivered to message listeners. Any calls to receive or message listeners that are in progress when stop is called, complete before the stop method returns.

When you no longer need a connection, close it by calling Connection.close to release its resources and help your application run more efficiently.

With a single connection to EAServer, the message service can send and receive multiple messages. Therefore, a JMS client usually needs only one connection.

Setting the client ID

A connection for a durable topic subscriber must have a client ID associated with it so that EAServer can uniquely identify a client if it disconnects and later reconnects.

The default client ID is the client's TCP/IP host name. To set the client ID to another value, you can either:

- Use the Management Console to set the client ID when you create the connection factory—see "Configuring JMS connection factories" on page 16, or
- Set it immediately after creating the connection and before performing any other action on the connection. After this point, attempting to set the client ID throws an IllegalStateException. This code fragment illustrates how to set a connection's client ID:

  ```
  myConnect.setClientID("Client ID String");
  ```

For more information about topic subscribers, see "Creating message consumers" on page 34.

ExceptionListener

To enable EAServer to asynchronously notify clients of serious connection problems, create and register an ExceptionListener. The javax.jms.ExceptionListener must implement this method:

```
void onException(JMSException exception);
```

To register a listener, call the Connection::setExceptionListener method, for example:

```
myConnect.setExceptionListener(MyExceptionListener);
```

If an exception occurs and a listener has been registered, EAServer calls the onException method and passes the JMSException, which describes the problem.

Creating sessions

Once a client has established a connection with EAServer, it must create one or more sessions. A session serves as a factory for creating message producers, message consumers, and temporary destinations.

Beginning in version 6.0, EAServer provides a session object that you can use for both message queues and topics.

To create a Session object, use a previously created Connection object as follows:

```
Session mySession = myConnect.createSession(true,
                            Session.AUTO_ACKNOWLEDGE);
```

When you create a session, set the first parameter to true if you want a **transacted session**. When you publish or send messages in a transacted session, a transaction begins automatically. Once a transacted session starts, all messages published or sent in the session become part of the transaction until the transaction is committed or rolled back. When a transaction is committed, all published or sent messages are delivered. If a transaction is rolled back, any messages that are produced in the session are destroyed, and any consumed messages are recovered. When a transacted session is committed or rolled back, the current transaction ends and the next transaction begins. See Chapter 2, “CORBA Component Life Cycles and Transaction Semantics,” in the *EAServer CORBA Components Guide* for more information about transactions.

Set the first parameter to false when you do not want a transacted session. If a client has an active transaction context, it can still achieve transactional behavior, even if it does not create a transacted session.

The second parameter indicates whether the message producer or the consumer will acknowledge messages. This parameter is valid only for nontransacted sessions. In transacted sessions, acknowledgment is determined by the outcome of the transaction.

Acknowledgment mode	Description
AUTO_ACKNOWLEDGE	The session automatically acknowledges messages.
CLIENT_ACKNOWLEDGE	The client explicitly acknowledges a message, which automatically acknowledges all messages delivered in the session.
DUPS_OK_ACKNOWLEDGE	EAServer implements this the same as AUTO_ACKNOWLEDGE.

Session.recover

To stop message delivery within a session and redeliver all the unacknowledged messages, you can use the Session.recover method. When you call recover for a session, the message service:

1 Stops message delivery within the session.

2 Marks all unacknowledged messages as “redelivered,” including those that have been delivered.

3 Restarts sending all unacknowledged messages, beginning with the oldest message.

The Session.recover method can be called only in a non-transacted session; it throws an IllegalStateException if it is called by a transacted session.

Session.close

The JMS provider can allocate resources on behalf of a session outside the JVM. Clients should use Session.close to close a session when it is no longer needed, to release its resources and help the application run more efficiently.

Temporary destinations

A temporary destination is generated automatically, and its scope is the connection in which it is created. A typical use for a temporary destination is as the destination to which a reply to a message should be sent. Specify this destination in the message header field JMSReplyTo.

Looking up destinations

JMS destinations are message queues and topics. Use the Management Console to create and configure JMS destinations—see “Message queues and topics” on page 8. However, you need not preconfigure message queues or topics, unless you want to set their properties to non-default values.

A JMS client can use the JMS initial context factory com.sybase.jms.client.InitialContectFactory to look up preconfigured message queues and topics by name. For example, if you create a message queue called “myMessageQueue” using the Management Console, you can look up the queue as follows, where *ctx* is the initial naming context:

```
Queue myQueue = (Queue) ctx.lookup(“myMessageQueue”);
```

To look up a message queue that has not been preconfigured, you can use either:

- JNDI to look up a name ending with “Queue”; for example:

  ```
  Queue myQueue = (Queue) ctx.lookup(“newQueue”);
  ```

  Look ups are not case sensitive.

- The javax.jms.Session.createQueue API; for example:

  ```
  Queue myQueue =
      (Queue) mySession.createQueue(“newQueue”)
  ```

  You can also use this API to look up preconfigured queues.

When you look up a message queue or topic that has not been preconfigured, the message service initializes an instance of the message queue or topic using default property values.

To look up message topics that have not been preconfigured, clients can use either a JNDI name ending with “Topic,” or use the javax.jms.Session.createTopic API. You can also use this API to look up preconfigured topics.

Creating message producers and consumers

Clients use message producers to send or publish messages, and message consumers to receive messages. To create message producers or consumers, you need a valid JMS session.

Creating message producers

Create one or more message producers for sending and publishing messages.

Beginning in version 6.0, EAServer provides a common message producer object that you can use for both sending and publishing messages.

The following code fragment creates a MessageProducer object called "mySender" using the previously created Session object *mySession*, and specifies the name of the message queue destination *myQueue*:

```
MessageProducer mySender =
   mySession.createProducer(myQueue);
```

When you create a message producer, you can specify default values for the following properties:

- DeliveryMode – the mode of message delivery can be either:
  - PERSISTENT – messages are delivered *once and only once*. Saving messages to a data store provides consistent, nonduplicated message delivery.
  - NON_PERSISTENT – messages are delivered *at most once*. Messages are not saved to a data store. If the JMS provider fails, messages can be lost, but they are not delivered more than once. This mode requires less system overhead.
- Priority – the priority of a message can range from 0 – 9, where 0 is the lowest priority. Typically, 0 – 4 are considered normal priority levels, and 5 – 9 are expedited priority levels.
- Time-to-live – the number of milliseconds, which together with the GMT, defines a message expiration time.

When you no longer need a message producer, call MessageProducer.close to release its resources.

Creating message consumers

Message consumers receive messages that are either sent to a queue or published.

Beginning in version 6.0, EAServer provides a message consumer object, which you can use for receiving messages that are either sent or published. In earlier versions of EAServer, two types of message consumers were required, one for receiving messages that are sent to a queue and one for subscribing to published messages.

The following code fragment uses a previously created Session object *mySession*, and creates a MessageConsumer to retrieve messages that are sent to the message queue *myQueue*:

```
MessageConsumer myConsumer =
   mySession.createConsumer(myQueue);
```

A MessageConsumer object that acts as a topic subscriber receives published messages and can be either durable or nondurable. A nondurable subscriber can only receive published messages while it is connected to EAServer. If you want guaranteed message delivery, make the subscriber durable. For example, if you create a durable subscription on a topic, EAServer saves all published messages for the topic in a database. If a durable subscriber is temporarily disconnected from EAServer, its messages are delivered when the subscriber reconnects. Messages are deleted from the database only after they are delivered to the durable subscriber.

This example illustrates how to create both durable and nondurable topic subscribers. In both cases, reference previously created Topic and Session objects:

```
// Create a durable subscriber

MessageConsumer subscriber =
   mySession.createDurableSubscriber(myTopic,
      “mySubscription”)

// Create a non-durable subscriber

MessageConsumer subscriber =
   mySession.createSubscriber(myTopic);
```

To remove a durable topic subscription, call the Session.unsubscribe method, and pass in the subscription name; for example:

```
mySession.unsubscribe(“subscriptionName”);
```

When you no longer need a message consumer, call MessageConsumer.close to release its resources.

Filtering messages using selectors

You can use selectors to specify which messages you want delivered to a message queue. Once you add a selector to a queue, the message service delivers only those messages whose message topic matches the selector. You can define message selectors in the Ant configuration file that defines the component—see “Message selectors” on page 23. You can also create message selectors programmatically. The following example illustrates how to create a message selector and use it when you are creating a new MessageConsumer:

```
// Create a selector to receive only text messages whose value
// property equals 100.

String mySelector = “value = 100 and Type = ‘TextMessage’”;

// Create a MessageConsumer for a queue using mySelector.

MessageConsumer receiver = mySession.createConsumer(myQueue, mySelector);
```

This code sample sends two messages to the message queue we just created. The properties of the first message match those of the message queue’s selector. The properties of the second message do not.

```
// Create and send a message whose properties match the message queue selector.

TextMessage textMsg = mySession.createTextMessage(“Text Message”);
textMsg.setIntProperty(“Value”, 100);
textMsg.setStringProperty(“Type”, “TextMessage”);
sender.send(textMsg);

// Create and send a Bytes message, whose value property equals 200.

BytesMessage bytesMsg = mySession.createBytesMessage();
bytesMsg.setIntProperty(“Value”, 200);
bytesMsg.setStringProperty(“Type”, “BytesMessage”);
sender.send(bytesMsg);
```

When messages are retrieved from the message queue, the text message is returned but the bytes message is not.

Sending and receiving messages

Creating messages

To create a message, you must first create an instance of a Session object. See "Creating sessions" on page 31 for details. The following sample code creates a JMS text messages using the session called *mySession*:

```
javax.jms.TextMessage myTextMsg =
   mySession.createTextMessage("Text message");
```

EAServer supports six message types that a message producer can send or publish. Table 3-1 describes the message types and the javax.jms.Session interface APIs used to create instances of each.

Table 3-1: JMS message types

Message type	Create message API	Comments
Plain	Session.createMessage	Creates a message without a message body.
Text	Session.createTextMessage	Creates a message that can contain a string in the message body.
Object	Session.createObjectMessage	Creates a message that can contain a serializable Java object in the message body.
Stream	Session.createStreamMessage	Creates a message that can contain a stream of Java primitives in the message body. Fill and read the message sequentially.
Map	Session.createMapMessage	Creates a message whose body can contain a set of name-value pairs where names are Strings and values are Java primitive types.
Bytes	Session.createBytesMessage	Creates a message that can contain a stream of uninterpreted bytes in the message body.

To improve interoperability with non-Java clients or components and to improve message receivers' ability to filter messages, Sybase recommends that you use either plain messages or text messages.

Message selectors allow you to filter messages based on text in the message properties. You cannot filter messages based on text in the message body.

For more information about message types and message properties, see the JMS 1.1 API documentation at http://java.sun.com/products/jms/docs.html.

Sending messages

To send a message, you must specify the destination message queue. The message service notifies listeners that are registered for the queue and the message remains in the queue until it is received and acknowledged.

Figure 3-2: Send message flow

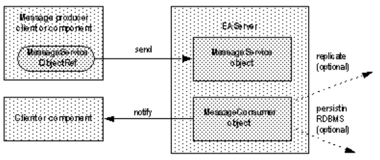


Figure 3-2 illustrates the message flow that occurs when a client or component sends a message.

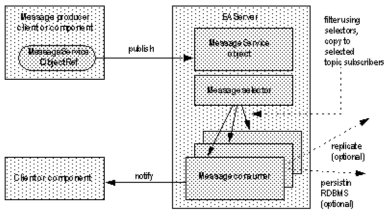
This example notifies a client of a completed order; it creates a new message, constructs the message text, and sends the message to the client’s queue:

```
public void notifyOrder(Session mySession,
                        Queue queue,
                        int orderNo,
                        String product)
{
      String text = "Order " + orderNo + " for product " + product +
      " was completed at " + time;

   MessageProducer sender = mySession.createProducer(myQueue);
   javax.jms.TextMessage textMsg = qSession.createTextMessage(text);

   textMsg.setStringProperty(“ProductDescription”, product);
   textMsg.setIntProperty(“OrderNumber”, orderNo);

   sender.send(textMsg);
}
```

Publishing messages

When you publish a message, a copy is sent to all topic subscribers that have a message selector registered with the specified topic. Figure 3-3 illustrates the message flow when a client or component publishes a message.

Figure 3-3: Publish message flow



This example publishes a message that notifies clients of changes in a stock value; it creates the message text, creates a MessageProducer and the message using the Session object, then publishes the message:

```
public void notifyStockValue(Session mySession,
                             Topic topic,
                             String stock,
                             double value)
{
   String text = time + ": The stock " + stock + " has value " + value;

   // Create the publisher and message objects.

   MessageProducer publisher = mySession.createProducer(topic);
   javax.jms.TextMessage textMsg = mySession.createTextMessage(text);


   // Publish a NON_PERSISTENT message with a priority of 9 and a
   // time-to-live of 5000 milliseconds (5 seconds)

   publisher.publish(textMsg, DeliveryMode.NON_PERSISTENT, 9, 5000);
}
```

To publish a persistent message using the default priority (4) and timeout (never expires) values, use this syntax:

```
publisher.publish(textMsg);
```

Receiving messages

You can receive messages either synchronously or asynchronously. To receive messages synchronously (get all the messages at one time), call the receive method for the message consumer. The following code samples illustrate how to receive all the messages from a queue, using three different timeout options:

```
// Get all the messages from the queue. If none exists, wait until a message
// arrives.

javax.jms.TextMessage queueTextMsg = (javax.jms.TextMessage)
   receiver.receive();

// Get all the messages from the queue. If none exists, wait 5000 milliseconds
//(5 seconds) or until a message arrives, whichever comes first.

javax.jms.TextMessage queueTextMsg =
   (javax.jms.TextMessage) receiver.receive(5000);

// Get all the messages from the queue. If none exists, return NULL.

javax.jms.TextMessage queueTextMsg =
   (javax.jms.TextMessage) receiver.receiveNoWait();
```

To receive messages asynchronously (as they are delivered), implement a message listener and install it on the message consumer, either a topic or a queue. See “Implementing and installing message listeners” on page 40.

Implementing and installing message listeners

Message listeners allow you to receive messages asynchronously. Once you have implemented a listener, install it on a message consumer. When a message is delivered to the message consumer, the listener can process the message or send it to other consumers.

The message service implements two types of JMS message listeners:

- Message-driven beans – see “Message-driven beans” on page 19.
- Client-side – you can associate a client-side JMS message listener with a message consumer programmatically.

A message listener implements the javax.jms.MessageListener interface, which contains only the onMessage method. This example illustrates the skeleton code for a message listener:

```
class QueueMessageListener implements MessageListener
{
   public void onMessage(javax.jms.Message msg)
   {
      // process message
   }
}
```

To install a client-side message listener, first create a message consumer (see “Creating message consumers” on page 34), then install the listener, using this syntax:

```
myConsumer.setMessageListener(new MessageListener());
```

Browsing messages

You can use the QueueBrowser interface to look at messages in a queue without removing them. You can browse through all the messages in a queue, or through a subset of the messages. To browse through a queue’s messages, create an instance of a QueueBrowser object using a previously created Session object. To create a browser for viewing all the messages in a queue, call createBrowser and pass the message queue:

```
QueueBrowser qbrowser =
   mySession.createBrowser(myQueue);
```

To create a browser for viewing a subset of the messages in a queue, call createBrowser and pass the message queue and a message selector string:

```
QueueBrowser qbrowser =
   mySession.createBrowser(myQueue, mySelector);
```

For information about message selectors, see “Filtering messages using selectors” on page 36.

Once you have access to the QueueBrowser object, call getEnumeration, which returns an Enumeration that allows you to view the messages in the order that they would be received:

```
java.util.Enumeration enum = qbrowser.getEnumeration();
```

Creating JMS providers

A JMS provider supports a messaging system by implementing the JMS API, and by providing other administrative and control functionality. You can define new JMS providers using either the Management Console or an XML configuration file.

You can run JMS application clients using the properties configured by the JMS provider. Properties set in the JMS provider definition take precedence over those set in the JMS client code or in Java system properties. See Chapter 4, “Creating Application Clients,” in the *EAServer Enterprise JavaBeans User’s Guide*.

❖ **Adding a JMS provider**

1. Start the Web console and connect to EAServer as described in Chapter 1, “Getting Started,” in the *System Administration Guide*.
2. Expand the Naming Providers folder and the JMS Providers folder beneath it.
3. Right-click the JMS Providers folder, and select Add. Complete the wizard to create the JMS provider.
4. Configure the JMS provider.

❖ **Configuring a JMS provider**

1. Expand the Naming Providers folder and the JMS Providers folder, then select the provider to configure.

   The JMS provider properties display on the following tabs:

   - General tab
   - JMS Settings tab
   - Pull Settings tab
   - Push Settings tab

   On each tab, click Apply to save your changes.

General tab

On the General tab, enter:

- **Use Resource Adapter** Select this option if MDBs listen on a JCA resource adapter. Using resource adapters allows EAServer to receive messages from third-party JMS providers. Resource adapters can also receive message types other than javax.jms.Message. The configuration property name is useResourceAdapter.
- **JNDI Initial Context Factory** The initial context factory for the JMS provider; for the default provider, the value is "com.sybase.jms.client.InitialContextFactory." The configuration property name is java.naming.factory.initial.
- **JNDI Provider URL** The URL for the name server. The default is "iiop://${host.name}:2000" (host.name represents the name of the machine where EAServer is running). The configuration property name is java.naming.provider.url.
- **JNDI Security Principal** A valid user name to access the name server; the default is "${user.name}." The configuration property name is java.naming.security.principal.
- **JNDI Security Credentials** A valid password for the user. he configuration property name is java.naming.security.credentials.
- **Idle Connection Timeout** The number of seconds before an idle connection times out; the default is 60 (1 minute). The configuration property name is idleConnectionTimeout.
- **Lookup Cache Timeout** The number of seconds cache entries remain active before they must be refreshed; the default is 600 (10 minutes). The configuration property name is lookupCacheTimeout.
- **Socket Timeout** The number of seconds a socket remains active; the default is 600 (10 minutes). The configuration property name is socketTimeout.
- **Enable Automatic Failover** If set to true, duplicate messages might be published. The configuration property name is automaticFailover.
- **Enable Data Compression** Select to enable data compression. The configuration property name is dataCompression.

JMS Settings tab

On the JMS Settings tab, enter:

- **JMS Queue Connection Factory** Enter the name of the class that provides queue connections; the default is QueueConnectionFactory. The configuration property name is queueConnectionFactory.
- **JMS Topic Connection Factory** Enter the name of the class that provides topic connections; the default is TopicConnectionFactory. The configuration property name is topicConnectionFactory.
- **JMS Connection Username** If required, a user name for a JMS connection. The configuration property name is jms.username.
- **JMS Connection Password** The password for the JMS connection user name. The configuration property name is jms.password.

Pull Settings tab

On the Pull Settings tab, enter:

- **Pull – Batch Size** The number of messages to pull into a message queue or topic from a queue or topic on a remote JMS provider, in a single batch. The default is 1. The configuration property name is pullBatchSize.
- **Pull – Receive Timeout** The number of seconds to pull messages into a message queue or topic from a queue or topic on a remote JMS provider, before timing out. The default is 600 (10 minutes). The configuration property name is pullReceiveTimeout.
- **Pull – Wait After Success** The number of seconds to wait after successfully pulling messages into a message queue or topic, before another pull operation is performed. The default is 0. The configuration property name is pullWaitAfterSuccess.
- **Pull – Wait After Failure** The number of seconds to wait after a failed attempt to pull messages into a message queue or topic, before trying again. The default is 60 (1 minute). The configuration property name is pullWaitAfterFailure.

Push Settings tab

On the Push Settings tab, enter:

- **Push – Batch Size** The number of messages to push onto one or more queues or topics on a remote JMS provider, in a single batch. The default is 1. The configuration property name is pushBatchSize.

- **Push – Receive Timeout** The number of seconds to push messages from a message queue or topic to one or more queues or topics on a remote JMS provider, before timing out. The default is 600 (10 minutes). The configuration property name is pushReceiveTimeout.
- **Push – Wait After Success** The number of seconds to wait after successfully pushing messages into a remote message queue or topic, before another push operation is performed. The default is 0. The configuration property name is pushWaitAfterSuccess.
- **Push – Wait After Failure** The number of seconds to wait after a failed attempt to push messages into a remote message queue or topic, before trying again. The default is 60 (1 minute). The configuration property name is pushWaitAfterFailure.

Index