Messaging Services User's Guide

Adaptive Server® Enterprise

Version 12.5.2

DOCUMENT ID: DC20154-01-1252-01

LAST REVISED: April 2004

Sybase, Inc., One Sybase Drive, Dublin, CA 94568.

Contents

About This Book

Audience

This book describes how to use the Real Time Messaging Services to integrate messaging services with all Adaptive Server® Enterprise database applications. This integration applies to any messaging service that interfaces with Java Message Service (JMS).

How to use this book

This book will assist you in installing, configuring, and using Real Time Messaging in Adaptive Server database applications. It includes these chapters:

- Chapter 1, “Introduction,” discusses messaging concepts, models, and format, and provides a short glossary of terms.
- Chapter 2, “Overview of Real Time Messaging Services” provides an overview of Real Time Messaging Services (RTMS) specific to Adaptive Server.
- Chapter 3, “Configuring Real Time Messaging Services” provides a procedure for configuring your system.
- Chapter 4, “Real Time Stored Procedures for Administration” documents the set of stored procedures provided to manage and administer Real Time Messaging Services.
- Chapter 5, “Real Time Data Functions” documents the SQL functions for managing and administering Real Time Messaging, and the general format of option strings.
- Chapter 6, “Common Topics in Real Time Messaging”describes global variables, properties, and XML documents that are used by several of the functions and procedures in Chapters 4 and 5.
- Chapter 7, “Transactional Behavior” describes transactional message requirements and behavior.
- Chapter 8, “Samples” provides code samples that illustrate messaging functionality.

Reference documents

- Java Message Service by “Java Technologies”(at http://java.sun.com/products/jms)

- TIBCO Enterprise JMS Message Bus by TIBCO Software, at (http://www.tibco.com).
- IBM MQSeries by IBM at (http://www.ibm.com/software/ts/mqseries)

Related documents

The following documents comprise the Sybase Adaptive Server Enterprise documentation set:

- The release bulletin for your platform – contains last-minute information that was too late to be included in the books.

  A more recent version of the release bulletin may be available on the World Wide Web. To check for critical product or document information that was added after the release of the product CD, use the Sybase Technical Library.
- The *Installation Guide* for your platform – describes installation, upgrade, and configuration procedures for all Adaptive Server and related Sybase products.
- *What's New in Adaptive Server Enterprise?* – describes the new features in Adaptive Server version 12.5.1, the system changes added to support those features, and the changes that may affect your existing applications.
- *ASE Replicator User's Guide* – describes how to use the ASE Replicator feature of Adaptive Server to implement basic replication from a primary server to one or more remote Adaptive Servers.
- *Component Integration Services User's Guide* – explains how to use the Adaptive Server Component Integration Services feature to connect remote Sybase and non-Sybase databases.
- *Configuring Adaptive Server Enterprise* for your platform – provides instructions for performing specific configuration tasks for Adaptive Server.
- *EJB Server User's Guide* – explains how to use EJB Server to deploy and execute Enterprise JavaBeans in Adaptive Server.
- *Error Messages and Troubleshooting Guide* – explains how to resolve frequently occurring error messages and describes solutions to system problems frequently encountered by users.
- *Full-Text Search Specialty Data Store User's Guide* – describes how to use the Full-Text Search feature with Verity to search Adaptive Server Enterprise data.
- *Glossary* – defines technical terms used in the Adaptive Server documentation.

- *Historical Server User's Guide* – describes how to use Historical Server to obtain performance information for SQL Server® and Adaptive Server.
- *jConnect for JDBC Programmer's Reference* – describes the jConnect™ for JDBC™ product and explains how to use it to access data stored in relational database management systems.
- *Job Scheduler User's Guide* – provides instructions on how to install and configure, and create and schedule jobs on a local or remote Adaptive Server using the command line or a graphical user interface (GUI).
- *Monitor Client Library Programmer's Guide* – describes how to write Monitor Client Library applications that access Adaptive Server performance data.
- *Monitor Server User's Guide* – describes how to use Monitor Server to obtain performance statistics from SQL Server and Adaptive Server.
- *Performance and Tuning Guide* – is a series of four books that explains how to tune Adaptive Server for maximum performance:
  - *Basics* – the basics for understanding and investigating performance questions in Adaptive Server.
  - *Locking* – describes how the various locking schemas can be used for improving performance in Adaptive Server.
  - *Optimizer and Abstract Plans* – describes how the optimizer processes queries and how abstract plans can be used to change some of the optimizer plans.
  - *Monitoring and Analyzing* – explains how statistics are obtained and used for monitoring and optimizing performance.
- *Quick Reference Guide* – provides a comprehensive listing of the names and syntax for commands, functions, system procedures, extended system procedures, datatypes, and utilities in a pocket-sized book.
- *Reference Manual* – is a series of four books that contains the following detailed Transact-SQL® information:
  - *Building Blocks* – Transact-SQL datatypes, functions, global variables, expressions, identifiers and wildcards, and reserved words.
  - *Commands* – Transact-SQL commands.
  - *Procedures* – Transact-SQL system procedures, catalog stored procedures, system extended stored procedures, and `dbcc` stored procedures.

- *Tables* – Transact-SQL system tables and dbcc tables.

- *System Administration Guide* – provides in-depth information about administering servers and databases. This manual includes instructions and guidelines for managing physical resources, security, user and system databases, and specifying character conversion, international language, and sort order settings.
- *System Tables Diagram* – illustrates system tables and their entity relationships in a poster format. Available only in print version.
- *Transact-SQL User's Guide* – documents Transact SQL, Sybase's enhanced version of the relational database language. This manual serves as a textbook for beginning users of the database management system. This manual also contains descriptions of the pubs2 and pubs3 sample databases.
- *Using Adaptive Server Distributed Transaction Management Features* – explains how to configure, use, and troubleshoot Adaptive Server DTM features in distributed transaction processing environments.
- *Using Sybase Failover in a High Availability System* – provides instructions for using Sybase's Failover to configure an Adaptive Server as a companion server in a high availability system.
- *Utility Guide* – documents the Adaptive Server utility programs, such as isql and bcp, which are executed at the operating system level.
- *Web Services User's Guide* – explains how to configure, use, and troubleshoot Web Services for Adaptive Server.
- *XA Interface Integration Guide for CICS, Encina, and TUXEDO* – provides instructions for using the Sybase DTM XA interface with X/Open XA transaction managers.
- *XML Services in Adaptive Server Enterprise* – describes the Sybase native XML processor and the Sybase Java-based XML support, introduces XML in the database, and documents the query and mapping functions that comprise XML Services.

Other sources of information

Use the Sybase Getting Started CD, the Sybase Technical Library CD, and the Technical Library 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 Technical Library CD. It is included with your software. To read or print documents on the Getting Started CD, you need Adobe Acrobat Reader (downloadable at no charge from the Adobe Web site, using a link provided on the CD).
- The Technical Library CD contains product manuals and is included with your software. The DynaText reader (included on the Technical Library CD) allows you to access technical information about your product in an easy-to-use format.

  Refer to the *Technical Library Installation Guide* in your documentation package for instructions on installing and starting the Technical Library.
- The Technical Library Product Manuals Web site is an HTML version of the Technical Library CD that you can access using a standard Web browser. In addition to product manuals, you will find links to EBFs/Updates, Technical Documents, Case Management, Solved Cases, newsgroups, and the Sybase Developer Network.

  To access the Technical Library Product Manuals Web site, go to Product Manuals at http://www.sybase.com/support/manuals/.

Sybase certifications on the Web

Technical documentation at the Sybase Web site is updated frequently.

v **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.

v **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

v **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. Enter user name and password information, if prompted (for existing Web accounts) or create a new account (a free service).
3. Select a product.
4. Specify a time frame and click Go.
5. Click the Info icon to display the EBF/Maintenance report, or click the product description to download the software.

Transact-SQL conventions

This book uses the same font and syntax conventions for Transact SQL as other Adaptive Server documents:

- Command names, command option names, utility names, utility flags, and other keywords are in Helvetica in paragraph text. For example:
  - select command
  - isql utility
  - -f flag
- Variables, or words that stand for values that you fill in, are in italics. For example:
  - *user_name*
  - *server_name*
- Code fragments are shown in a monospace font.Variables in code fragments (that is, words that stand for values that you fill in) are italicized. For example:

```
Connection con = DriverManager.getConnection
('jdbc:sybase:Tds:host:port', props);
```

- You can disregard case when typing Transact SQL keywords. For example, SELECT, Select, and select are the same.

Additional conventions for syntax statements in this manual are described in Table 1. Examples illustrating each convention can be found in the *System Administration Guide*.

Table 1: Syntax statement conventions

Key	Definition
{ }	Curly braces indicate that you choose at least one of the enclosed options. Do not include braces in your option.
[ ]	Brackets mean choosing one or more of the enclosed options is optional. Do not include brackets in your option.
( )	Parentheses are to be typed as part of the command.
|	The vertical bar means you may select only one of the options shown.
,	The comma means you may choose as many of the options shown as you like, separating your choices with commas to be typed as part of the command.

If you need help

Each Sybase installation that has purchased a support contract has one or more designated people who are authorized to contact Sybase Technical Support. If you cannot resolve a problem using the manuals or online help, please have the designated person contact Sybase Technical Support or the Sybase subsidiary in your area.

CHAPTER 1

Introduction

Although this book assumes a basic knowledge of messaging systems in database management, this chapter introduces some basic message concepts and models, and provides a short glossary of terms.

Most of the discussion concerns the new aspects of the messaging available in Adaptive Server. This set of messaging services functionality is also referred to in the document as Real Time Messaging Services (RTMS).

RTMS messaging concepts

Messaging is the exchange of information by two or more software applications. A message is a self-contained package of information.

Many Adaptive Server customers use messaging and queuing, or publishing and subscription systems in their own application environments. These applications are called message-oriented middleware. Often the same application combines database operations with messaging operations.

Real Time Messaging Services (RTMS), simplifies the development of such applications, using both Adaptive Server and the TIBCO™ Java Messaging Service (JMS).

Messaging systems allow senders and receivers to be decoupled. Not all components need to be up, running, and connected for operation at all times. A messaging system can be asynchronous, in that an application can send messages without requiring receiving applications to be up and running.

JMS is an API that defines the way in which clients communicate with message providers. The message sender and the message receiver both act as clients to the message provider.

Messaging systems are provided by message providers. The messaging providercan implement architecture that is either centralized or decentralized, or sometimes a hybrid of the two.

RTMS provides an easy way to perform messaging operations within SQL statements, using built-in functions.

Real Time Messaging Services provide a way to capture transactions (data changes) in an Adaptive Server Enterprise database and deliver them as events to external applications through a JMS message bus, provided by TIBCO Enterprise™ for JMS.

Automatic decisions in Real Time

In managing a database it is sometimes necessary to make automated decisions in real time, in response to specific events. Real time means that the database can make decisions regarding events at the same time the events occur, rather than simply queuing the events. An event, such as a change in a record, must be evaluated in conjunction with other changes, and the most efficient response chosen. This means that effective decision support systems need:

- Low latency, enabling real-time enterprise
- An automated system that describes events and the data relating to them
- A technology to reduce the cost of applications that deliver low latency

These business needs are addressed by Sybase RTDS and RTMS, using the TIBCO JMS message bus.

Messaging models

JMS defines two messaging models:

- Publish-and-subscribe
- Point-to-point

Publish-and-subscribe

The publish-and-subscribe model is a one-to-many model. In this type of messaging model the message initiates with the sender, and the message producer does not depend on message consumers receiving the message. The application sending the message is called the “message producer,” and the applications receiving the message are called “message consumers.” Message consumers establish subscriptions to register an interest in messages sent to a topic. A **topic** is the destination of this message model.

There are two types of subscriptions you can establish in this model:

- Durable
- Nondurable

A durable subscription retains messages for the message consumer even when the message consumer application is not connected. The message provider, not Adaptive Server, retains the message.

A nondurable subscription retains messages only when consumer applications are connected to the message provider.

Point-to-point

The point-to-point model is a one-to-one model, in the sense that any message sent, by an application called a “message sender,” can be read only by one receiving application, called a “message receiver.” The destination of a point-to-point message is a **queue**. A queue may contain more than one active message receiver, but the messaging providerensures that the message is delivered to only one message receiver.

Message format

The JMS message format comprises three parts:

- Message header
- Message body
- Message properties

Message header

The header contains information specified by the JMS standard. Most of this information is automatically assigned by the message provider.

Message body

The message body is the application data that client applications exchange. JMS defines structured message types, such as stream and map, and unstructured message types, such as text, byte, and object.

Message properties

Message properties are user-defined additional headers that you can include with the message. Message properties have types, and these types define application-specific information that message consumers can use later, to select the messages that interest them. Message property types are Java native types: int, float, or String (class).

Message selectors

Message selectors provide a way for message consumers to filter the stream of messages and select the messages that interest them. These filters apply criteria that reference message properties and their values. The message selector is a SQL 92 where clause.

Glossary

The terms defined here are used throughout this document.

Term	Definition
Durable subscription	A subscription that retains messages while the client is not connected.
JMS	Java Message Service.
Messaging client	A program that produces or consumes messages.
MOM	Message-oriented middleware.
Nondurable subscription	A subscription that retains messages only while the client is connected.
Queue	A domain for point-to-point messaging.
Service provider	Message provider. For instance, TIBCO JMS is a service provider, called a messaging providerin this document.
Subscription	A domain for publishing or consuming one-to-many messaging.
Topic	Similar to queues, but used for one-to-many messaging.
URI	Uniform Resource Identifier, a string that identifies a resource on the web. See http://www.w3.org/Addressing/
URL	Uniform Resource Locator, another term for Uniform Resource Identifier.

CHAPTER 2

Overview of Real Time Messaging Services

This chapter provides an overview of Real Time Messaging Services (RTMS) specific to Adaptive Server, which allows you to use Adaptive Server as a client of the message provider. You can send messages to or retrieve messages from the messaging providerby using the Transact-SQL interface.

Sending and receiving messages from a queue

Using the built-in functions msgsend and msgrecv, Transact-SQL applications can send messages to a JMS queue or read messages from a JMS queue.

A message body, or payload, can be constructed using application logic, or it can contain character or binary data directly from relational tables.

You can construct the values of message properties from relational data or from application logic, and specify message properties when you send the message.

Messages read from the JMS queue can be processed by the application logic, or directly inserted into relational tables.To filter out only messages of interest when executing the read operation, specify a message selector.

Message properties in read messages can be individually processed by the application logic. For more information about message properties, see msgsend on page 23.

Publishing and consuming messages from a topic

Using the built-in functions msgpublish and msgconsume, Transact-SQL applications can publish messages to, or consume messages from, a JMS topic.

First, you must register a subscription, using sp_msgadmin 'register'. Registering a subscription creates a name that msgpublish, msgconsume, msgsubscribe, and msgunsubscribe can reference. A subscription can be registered as a durable or nondurable subscription, and you can specify a message selector to control the messages that come in, ensuring that only messages of interest are read.

You can use msgsubscribe to tell the JMS provider to hold messages until the application logic is ready to process them. Use msgunsubscribe to tell the JMS provider that the application is no longer interested in messages on this subscription. msgunsubscribe can also delete durable subscriptions from the JMS provider.

Message properties in read messages can be individually processed by the application logic.

Working with message properties

When a message is read, the message properties can be processed by Transact-SQL application logic, using built-in SQL functions. These functions return:

- The name of the nth property
- The value of a named property
- The type of a named property
- The number of properties
- A list of the properties

These built-in functions allow application logic to make processing decisions during runtime, based on the value of the message properties. The built-in functions are:

- msgproplist
- msgpropname
- msgpropvalue

- msgproptype
- msgpropcount

Previewing the messaging interface

The examples in this section provide a brief preview of the Transact-SQL messaging interface.

Example 1

This example sends a message to a queue:

```
select msgsend('hello world',
     'tibco_jms:tcp://my_jms_host:7222?queue=queue.sample'
     MESSAGE PROPERTY 'city=Detroit')
```

Example 2

This example reads a message from a queue, with and without a filter:

```
select msgrecv('tibco_jms:tcp://my_jms_host:7222?queue=queue.sample')
select msgrecv
     ('tibco_jms:tcp://my_jms_host:7222?queue=queue.sample'
     MESSAGE SELECTOR 'city=''Detroit''')
```

Example 3

This example publishes a message to a topic:

```
sp_msgadmin register, subscription,sub1,
     'tibco_jms:tcp://my_jms_host:7222?topic=topic.sample'
select msgpublish
     ('hello world', 'sub1' MESSAGE PROPERTY 'city=Boston')
```

Example 4

This example consumes a message from a topic:

```
select msgconsume('sub1')
```

Example 5

This example illustrates working with properties:

```
select msgconsume('sub1')
declare @pcount integer
declare @curr integer
declare @pname varchar(100)
select @curr=1
select @pcount = msgpropcount()
while(@curr<=@pcount)
begin
      select @pname=msgpropname(@curr)
      select msgproptype(@pname)
      select msgpropvalue(@pname)
      select @curr=@curr+1
end
```

CHAPTER 3

Configuring Real Time Messaging Services

This chapter has instructions for installing and configuring Real Time Messaging Services (RTMS) in Adaptive Server Enterprise.

Configuring RTMS

To install Sybase RTMS, just install Adaptive Server. Installing RTMS requires an ASE_MESSAGING license.

v **Configuring RTMS**

1 Install the Real Time Messaging Services stored procedures:

```
isql -i$SYBASE/$SYBASE_ASE/scripts/installmsgsvss
```

2 Grant messaging_role to the appropriate Adaptive Server logins:

```
grant role messaging_role to <login>
```

3 Configure the server to use Real Time Messaging Services:

```
sp_configure 'enable real time messaging', 1
```

4 Increase memory configuration for Real Time Messaging Services:

```
sp_configure 'messaging memory', <# of pages>
```

The default value is 400 pages. Increase this value if your application requires more memory.

5 If you have not already, add the local server and reboot:

```
sp_addserver <local server name>, local
```

You must reboot Adaptive Server for this command to take effect.

CHAPTER 4

Real Time Stored Procedures for Administration

This chapter describes a stored procedure, sp_msgadmin and its options, which you can use to manage and administer Real Time Messaging Services (RTMS).

Note sp_msgadmin and its options do not configure or administer the underlying message provider. For instance, you must still create, delete, and access queues and topics at the messaging provider level.

Each procedure in this chapteris a variant of sp_msgadmin. Parameters and examples are documented for each procedure.

The Usage section of sp_msgadmin is common to all of the variants of sp_msgadmin.

sp_msgadmin

Description

This stored procedure configures and administers messaging-related information.

Syntax

```
sp_msgadmin

sp_msgadmin ‘help’,
sp_msgadmin ‘help’, ‘list’
sp_msgadmin ‘help’, ‘register'
```

sp_msgadmin ‘help’, ‘default’
sp_msgadmin ‘help’, ‘remove’
sp_msgadmin 'list', 'provider' [, *provider_name*]
sp_msgadmin 'list’, ’login’ [, *provider_name*, [*login_name*],
sp_msgadmin 'list', 'subscription' [, *subscription_name*]
sp_msgadmin 'register', 'provider', *provider_name*, *provider_class*, messaging_provider_URL
sp_msgadmin 'register', 'login', *provider_name*, *local_login*, *provider_login*, *provider_password* [, *role_name*]
sp_msgadmin 'register’, 'subscription', *subscription_name*, endpoint [, selector [, *delivery_option* [, *durable_name*, *client_id*]]]
sp_msgadmin 'default’, 'login', *provider_name*, *provider_login*, *provider_password*
sp_msgadmin 'remove', ’provider', *provider_name*
sp_msgadmin 'remove', 'login', provider_name, *local_login* [, role]
sp_msgadmin 'remove', 'subscription', *subscription_name*

Examples

```
sp_msgadmin ’register’,’provider’,’one_jms_provider’,
     ’TIBCO_JMS’,’tcp://my_jms_host:7222’

sp_msgadmin ’default’, ’login’, ’one_jms_provider’, ’loginsa’,
     ’abcdef123456’
```

Usage

sp_msgadmin cannot be used inside a transaction.

Permissions

The permissions required depend on the subcommand.

sp_msgadmin 'help'

Description

Provides syntax information about this stored procedure or about particular parameters.

Syntax

sp_msgadmin 'help'
sp_msgadmin 'help', 'list'
sp_msgadmin 'help', ’register’
sp_msgadmin 'help', ’default’
sp_msgadmin 'help’, ’remove’

Parameters

’list’

Lists syntax information about message providers, logins, or subscriptions.

'register'
Provides stored procedure syntax to register a message provider, login, or subscription.

'default'
Lists the syntax to specify the default login for a specified message provider.

'remove'
Lists the stored procedure syntax to remove a message provider, login, or subscription.

Examples

Describes the syntax for sp_msgadmin 'list':

```
sp_msgadmin 'help', 'list'
```

Usage

See the Usage section under sp_msgadmin on page 13.

Permissions

No special permissions are required.

sp_msgadmin 'list'

Description

Lists information for message provider, logins, or subscriptions. If you specify the *provider_name*, *login_name*, or *subscription_name*, information is listed for that particular provider, login, or subscription. Otherwise, information is listed for all of them.

Syntax

sp_msgadmin 'list', 'provider' [, *provider_name*]
sp_msgadmin 'list' , login', [, *provider_name* [, *login_name*]]
sp_msgadmin 'list' , 'subscription' [, *subscription_name*]

Parameters

'provider'
Lists information about a particular messaging provider or about all message providers.

'login'
Lists information about a particular messaging provider login mapping or about all messaging provider logins.

'subscription'
Lists information about a particular subscription or about all subscriptions.

provider_name
An alias referring to the message provider, which can be as many as 30 characters in length.

login_name
A login name.

subscription_name
A subscription name.

Examples

Example 1 Lists the details for the “my_jms_provider” message provider:

```
sp_msgadmin ’list’, ’provider’, ’my_jms_provider’
```

Example 2 Lists the details for the user with a login of “loginsa”:

```
sp_msgadmin ’list’, ’login’, ’my_jms_provider’,’loginsa’
```

Example 3 Lists the details for subscription “subscription_1”.

```
sp_msgadmin ’list’, ’subscription’,’subscription_1’
```

Usage

See the Usage section under sp_msgadmin on page 13.

Permissions

No special permissions are required.

sp_msgadmin 'register'

Description

Registers a specified message provider, login, or subscrption.

Syntax

sp_msgadmin 'register', ’provider’, *provider_name*, *provider_class*, *messaging_provider_URL*
sp_msgadmin 'register', ’login’, *provider_name*, *local_login*, *provider_login*, *provider_password* [, *role_name*]
sp_msgadmin 'register', 'subscription’, *subscription_name*, *endpoint* [, *selector* [, *delivery_option* [, *durable_name*, *client_id*]]]

Parameters

‘provider’
Registers a message provider.

provider_name
An alias referring to the messaging provider you are adding, which can be as many as 30 characters in length. In the case of register provider, this is an alias for messaging_provider. In the case of register login, this is the name of a previously registered provider.

provider_class
The class of the messaging provider you are adding; for example, “TIBCO_JMS”.

messaging_provider_URL
The URL of the messaging provider you are registering.

'login'
Registers a login mapping.

local_login
An Adaptive Server login that maps to the local login.

provider_login
The login name of the messaging provider that *local_login* maps to when connecting to the message provider.

provider_password
The messaging provider password of the *provider_login*.

role_name
A SQL role name. If you specify a *role_name*, the *local_login* is ignored, and the *provider_login* and *provider_password* apply to the *role_name*.

'subscription'
Registers a subscription.

subscription_name
The name of the subscription you are registering.

end_point
The topic to which the subscription is addressed. See the description of *end_points* in msgsend on page 23.

selector
A message filter that allows a client to select messages of interest. See the description of filters in msgrecv on page 28.

delivery_option
One of "local," "nonlocal," or null. If the value is null it is assumed to be local. If you specify "local," a given SQL session can consume messages that it publishes. If you specify "nonlocal," a specified SQL session cannot consume messages that it publishes.

durable_name
A character string value. See the description of *client_id*, below.

client_id

The identification used by the messaging provider to identify the subscription as durable. *client_id* is a character string value. If either *client_id* or *durable_name* is specified, the other must also be specified, and the subscription is a durable subscription. Otherwise, it is a nondurable subscription.

The *client_id* and *durable_name* combination identifies durable subscriptions with the message provider, and must be unique. No two subscriptions can have the same *client_id* and *durable_name*.

client_id uniqueness extends across the messaging provider. JMS allows a particular *client_id* to be connected only once at any given time. For instance, if one application already has a durable subscription using a specified *client_id*, the *client_id* specified by another application should not be the same, if the applications are to be connected at the same time.

A durable subscription exists even when the client is not connected. The messaging provider saves messages that arrive even while the client is not connected.

A nondurable subscription exists only while the client is connected. The messaging provider discards messages that arrive while the client is not connected.

In the following example, sp_msgadmin registers a durable subscription named *durable_sub1*, Then sp_msgadmin 'list' displays information about the new subscription.

```
sp_msgadmin
     'register', 'subscription','durable_sub1',
     'my_jms_provider?topic=topic.sample',
     null, null, 'durable1', 'client1'
sp_msgadmin 'list', 'subscription', 'durable_sub1'
```

Examples

Example 1 Registers the messaging provider “my_jms_provider”:

```
sp_msgadmin 'register', 'provider', 'my_jms_provider', 'TIBCO_JMS',
     'tcp://10.23.233.32:4823'
```

Example 2 Registers the login “ase_login1”, using messaging provider login “jms_user1”, and messaging provider name “my_jms_provider”:

```
sp_msgadmin 'register', 'login', 'my_jms_provider', 'ase_login1',
     'jms_user1', 'jms_user1_password'
```

Example 3 Registers a login with the messaging provider login "jms_user1", and a specified password used for all Adaptive Server logins that have sa_role permissions:

```
sp_msgadmin 'register', 'login', 'my_jms_provider', null,
     'jms_user1', 'jms_user1_password', 'sa_role'
```

Example 4 Registers a login using the messaging provider login "jms_user1", and a specified password used for all unmapped Adaptive Server logins:

```
sp_msgadmin 'register', 'login', 'my_jms_provider', null, 'jms_user1',
     'jms_user1_password'
```

Example 5 Registers "subscription_1", a nondurable subscription.

```
sp_msgadmin 'register', 'subscription', 'subscription_1',
     'my_jms_provider?topic=topic.sample'
```

Note "my_jms_provider" must be a provider that you have previously registered, using sp_msgadmin register, provider.

Usage

- When a login name is used to connect to the message provider, login names are resolved in the following order:
  - a Explict login names and passwords, specified in the end point, if provided.
  - b Explicit login mapping for the current Adaptive Server login.
  - c The default login name and password for the message provider, and the role corresponding to the Adaptive Server login.
  - d The default login name and password for the message provider, with no specific role association.
  - e Null login name and password if none of the above apply.
- You can modify the login mapping between the Adaptive Server login and the messaging provider login only by removing and reregistering it with a different set of mappings.
- See sp_msgadmin on page 13 for usage common to the variants of sp_msgadmin.

Permissions

You must have messaging_role and sso_role permissions to issue sp_msgadmim 'register'.

sp_msgadmin 'default'

Description

Specifies a default login and password for the specified message provider.

Syntax

sp_msgadmin ’default’, ’login’, *provider_name*, *provider_login*, *provider_password*

Parameters

‘default’
Specifies a default.

‘login’
Specifies a default login.

provider_name
The name of the messaging provider to remove, or the name of the message provideer from which the specified login is removed.This can be as many as 30 characters long.

provider_login
The login the provider uses as the default login when sending or receiving messages from the messaging provider specified by *provider_name*.

provider_password
The password used for the *provider_login*.

Examples

Specifies the default login that applies to all unmapped Adaptive Server logins, when using a specified messaging provider for either sending or receiving. You must first register the *provider_name* by calling sp_msgadmin 'register', 'provider'.

```
sp_msgadmin ’default’, ’login’, ’my_jms_provider’,
     ’jms_user1’, ’jms_user1_password’
```

Usage

- You can check the default login by entering:

```
sp_msgadmin ’list’, ’login’, ’my_jms_provider’
```

- You can remove the default login by entering’

```
sp_msgadmin ’remove’, ’login’, ’my_jms_provider'
```

See sp_msgadmin on page 13 for usage common to the variants of sp_msgadmin.

Permissions

You must have messaging_role and sso_role permissions to issue sp_msgadmin ’default’.

sp_msgadmin 'remove'

Description

Removes the specified message provider, login, or subscription.

Syntax

sp_msgadmin 'remove', 'provider', *provider_name*
sp_msgadmin 'remove', 'login', *provider_name*, *local_login* [, *role_name*]
sp_msgadmin 'remove', "subscription", *subscription_name*

Parameters

'provider'

Removes a messaging provider previously defined by this call:

```
sp_msgadmin 'register', 'provider', provider_name
```

provider_name

The name of the messaging provider from which the specified login is removed.

'login'

Removes the mapping previously created between an Adaptive Server login and a service provider login, defined by this call:

```
sp_msgadmin 'register', 'login', local_login,...
```

local_login

The mapping to the Adaptive Server login that you are removing.

role_name

The name of the SQL role specified.

'subscription'

Removes a subscription previously created by this call:

```
sp_msgadmin 'register', 'subscription', subscription_name, ...
```

subscription_name

The name of the subscription to remove.

Examples

Example 1 Removes the messaging provider "my_jms_provider":

```
sp_msgadmin 'remove', 'provider', 'my_jms_provider'
```

Example 2 Removes the Adaptive Server login "ase_login1"associated with the messaging provider "my_jms_provider":

```
sp_msgadmin 'remove', 'login', 'my_jms_provider','ase_login1'
```

Example 3 Removes the default login, indicated by a null login parameter:

```
sp_msgadmin 'remove', 'login', 'my_jms_provider',null
```

Example 4 Removes all logins for role "sa_role" on "my_jms_provider":

```
sp_msgadmin 'remove', 'login', 'my_jms_provider',null, 'sa_role'
```

Example 5 Removes the subscription “subscription_1”:

```
sp_msgadmin 'remove', 'subscription', 'subscription_1'
```

Usage

- Removing a messaging provider does not affect messages that are in transit (that is, messages that are in the process of being sent or received) to this message provider.
- sp_msgadmin 'remove' does not affect any current connections to the message provider. This means that if a message provider, login, or default is removed while there is a current connection to the specified message provider, the connection is not affected. However, Sybase does not recommend this practice.
- See sp_msgadmin on page 13 for usage common to the variants of sp_msgadmin.
- You must specify *local_login* as null if you specify *role_name*.

Permissions

You must have messaging_role and sso_role permissions to issue sp_admin 'remove'.

CHAPTER 5

Real Time Data Functions

This chapter describes the SQL functions for administering Real Time Messaging, and the general format of option strings. See Table 6-2 to see Adaptive Server-specific message properties.

The SQL functions in this chapter:

- Send and receive messages to queues
- Publish, subscribe, and consume messages relating to message topics
- Handle message properties

msgsend

Description

Provides a SQL interface to send messages to different service endpoints. The endpoints are of type queue.

Syntax

message_send_call ::=
    msgsend (*message_body*, *end_point* [*options_and_properties*])

options_and_properties ::=
    [*option_clause*] [*properties_clause*]
option_clause::= [,] option *option_string*
properties_clause ::= [,] message property *option_string*
message_body :=
    scalar_expression
    | (*select_for_xml*)
end_point := *basic_character_expression*

Parameters

message_body

The message you are sending. The message body can contain any string of characters. It can be binary data, character data, or SQLX data.

end_point

The queue to which a message is addressed. It is a *basic_character_expression* whose runtime value is a *service_provider_uri*.

option_string

The general format of an *option_string* is specified in the section option_string—general format.

The options you can specify for msgsend are in Table 5-1:

Table 5-1: Options for msgsend

Option	Values	Default	Comments
schema	no yes "user_schema"	no	• *user_schema* is a user-supplied schema describing the *message_body*. • “no” means that no schema is generated and sent out as part of the message. • “yes” means that Adaptive Server generates an XML schema for the message. “yes” is only meaningful in a *message_body* that uses the parameter *select_for_xml*. *select_for_xml* generates a SQLX-formatted representation of the SQL result set. The generated XML schema is a SQLX-formatted schema that describes the result set document. The schema is included in the message as the ASE_MSGBODY_SCHEMA property.
type	text | bytes	text	The type of message to send.

properties_clause

Is either an *option_string* or one of the options listed in Table 5-2. The options described in Table 5-2 are set as a property in the message header or message properties, as indicated in the disposition column of the table. The option value is the property value.

Property names are case sensitive.

If you use a property not listed in Table 5-2, it is set as a property in the message properties of the message sent.

Table 5-2: Option values for properties of msgsend

Option	Values	Default	Disposition	Comments
mode	persistent or nonpersistent	persistent	header	If the mode is persistent, the message is backed by the JMS provider, using stable storage. If the messaging provider crashes before the message can be consumed, the message is lost, unless mode is set to persistent. If the mode is nonpersistent, and the messaging provider crashes, you may lose a message before it reaches the desired destination.
ttl	0 - (2^{63}- 1)	0	header	ttl refers to time-to-live on the messaging bus. Adaptive Server is not affected by this. Expiry information, which is the duration of time during which the message is valid, in milliseconds. For instance, 60 means that the life of the message is 60 milliseconds. A value of 0 means that the message does not expire.
priority	1 to 10	4	header	The behavior of priority is controlled by the underlying message bus. The values mentioned here apply to TIBCO_JMS. Priorities from 0 to 4 are normal; priorities from 5 to 9 are expedited.
correlation	string	none	header	Client applications set correlation IDs to link messages together. Adaptive Server sets the correlation ID the application specifies.
replyqueue replytopic	A string containing a *queue_name* A string containing a *topic_name*	none none	header header	The value of *queue_name* or *topic_name* must be syb_temp. The type of the temporary destination, queue or topic, depends on whether you specify replyqueue or replytopic. Only the option listed last is used. Adaptive Server creates a temporary destination and sends information related to the newly created temporary destination as a part of the header information.

scalar_expression
If a message is a SQL *scalar_expression*, it can be of any datatype.

If the type option is not specified, the message type is text if the *scalar_expression* evaluates to a character datatype; otherwise the message type is bytes.

If the datatype of the *scalar_expression* is not character, it is converted to varbinary using the normal SQL rules for implicit conversion. The binary value of the datatype is included in the message according to the byte ordering of the host machine.

basic_character_expression
A Transact-SQL query expression whose datatype is char, varchar, or java.lang.String.

select_for_xml
A select expression that specifies a `for xml` clause.

Examples

Example 1 Sends the message "hello" to the specified endpoint:

```
select msgsend('Hello','my_jms_provider?queue=queue.sample,'
     +'user=jms_user1,password=jms_user1_password')
```

Example 2

Sends the message "Hello Messaging World!" to the specified endpoint:

```
declare @mymsg varchar (255)
set @mymsg = 'Hello Messaging World!'
select msgsend(@mymsg,
     +'my_jms_provider?queue=queue.sample,user=jms_user1,'
     +'password=jms_user1_password')
```

Example 3 Sends a message whose body is a SQLX-formatted representation of the SQL result set, returned by the SQL query to the specified endpoint:

```
select msgsend ((select * from pubs2..publishers FOR XML),
     'tibco_jms:tcp://my_jms_host:7222?queue=queue.sample,'
     +'user=jms_user1,password=jms_user1_password')
```

Example 4 Sets two properties and generates an XML schema for the message:

```
select msgsend
((select pub_name from pubs2..publishers where pub_id = '1389' FOR XML),
     my_jms_provider?queue=queue.sample',
     MESSAGE PROPERTY 'priority=6, correlationID=MSG_001',
OPTION 'schema=yes')
```

Example 5 Shows user-specified values for message properties:

```
select msgsend ( 'hello', 'my_jms_provider?queue=queue.sample'
     MESSAGE PROPERTY 'ttl=30,category=5, rate=0.57, rank=''top'',
     priority=6')
```

ttl and priority are internally set as header properties. category, rate, and rank are set as user-specified properties in the message properties.

Usage

- The result of a msgsend call is a varchar string. If the message succeeds, the returned value is the message ID. If the message is not sent, the return value is null.
- In a *message_body* that is a *select_for_xml* parameter, *select_for_xml* generates a SQLX-formatted representation of the SQL result set.
- You can specify *select_for_xml* only if Adaptive Server is configured for the native XML feature. You can reference *select_for_xml* only as a scalar expression from a msgsend call.
- You must surround *select_for_xml* with parentheses, as shown in the Syntax section.
- The following restrictions apply to a runtime format for service_provider_uri:

```
service_provider_uri ::=
     provider_name ?destination [,user=username, password=password]
provider_name ::=
     local_name | full_name
local_name ::= identifier
full_name ::=
     service_provider_class:service_provider_url
```

  - The *local_name* is a provider identifier, previously registered in a call to sp_msgadmin 'register', 'provider', which is shorthand for the *full_name* specified in that call.
  - The service_provider_class currently supported is TIBCO_JMS.
  - The service_provider_url has the form “tcp://hostname:port”. The host name can be a name or an IP address.
  - A service_provider_url cannot have spaces.
- If the destination has the form `queue=queue_name`, the message is sent to this queue.

- The service_provider_class and the words “user”, and “password” are case insensitive. local_name, hostname, port, queue_name, user_name, and password parameters are case-sensitive.
- You can set message properties specific to Adaptive Server according to Table 6-2.
- Option string usage in msgsend:
  - Empty option strings are ignored.
  - You can separate option strings with commas or white space (there is no limit on the amount of white space before first option, after the last option, between options, and surrounding the equal signs).
  - Quoted strings are formed according to SQL conventions for embedded quotation marks.
  - If you specify multiple options with the same name, only the option listed last is processed. For example, in the following statement, only the value 7 is used or validated for `'priority'`; the other values are ignored:

```
select msgsend( 'Hello Messaging World!',
     'my_jms_provider?queue=queue.sample',
     MESSAGE PROPERTY 'priority=''high'', priority=yes, priority=7')
```

- After you execute msgsend, the values of the global variables are set with information for that call. For more details, see “Message-related global variables” on page 47.
- Use single apostrophes ('), not double quotation marks ("), around quoted option or property values.

Note msgsend also allows messages to be sent to a topic, if you specify `topic=topic_name` as the destination. Sybase does not recommend this practice, as it may cause unexpected behavior.

msgrecv

Description

Provides a SQL interface to receive messages from different service endpoints, which must be queues.

msgrecv receives a message from the specified *service_provider* and *service_destination*, and returns that message. The value returned is the message body returned by the service provider, converted to the specified return type.

Syntax

msgrecv_call ::=
msgrecv (*end_point options_filter_and_returns*)
options_filters_and_return ::=
[*option_clause*] [*filter_clause*] [*returns_clause*]
option_clause::= [,] option *option_string*
filter_clause::= [,] message selector *message_filter*
message_filter::=*basic_character_expression*
returns_clause ::= [,] returns *sql_type*
end_point := *basic_character_expression*
sql_type ::=
varchar(*integer*) | java.lang.String | text
| varbinary(*integer*) | image

Parameters

end_point
A *basic_character_expression* whose runtime value is a *service_provider_uri*. The destination of a message.

option_string
The general format of the option_string is specified in option_string—general format. The options for msgrecv are described in Table 5-3.

filter_clause
Passes a *message_filter* directly to a specified message provider, which determines its use.

message_filter
A filter parameter and *basic_character_expression*. The filter value is passed directly to the message provider. Its use depends on the message provider. See the Usage section below for a discussion of message filters.

basic_character_expression
A SQL query expression whose datatype is char, varchar, or java.lang.String.

returns_clause
The datatype that you want returned.

If you do not specify a *returns_clause*, the default is varchar(16384).

If you specify a *returns_clause* of type varbinary or image, the data is returned in the byte ordering of the message.

sql_type
The SQL datatype. The legal SQL datatypes are:

- varchar(n)
- text
- java.lang.String
- varbinary(n)
- image
- univarchar(n)

msgrecv
Receives a message from the specified *service_provider* and *service_destination*, and returns that message. The value returned is the message body returned by the service provider, converted to the specified return type. msgrecv option values are shown in Table 5-3 on page 30.

Table 5-3: msgrecv option values

Option	Values	Default	Comment
timeout	-1, 0 - (2^{31}- 1)	-1	By default, msgrecv is a blocking command, which blocks the message until it reads the next message from the message bus. If timeout is not -1, msgrecv returns null when the timeout interval lapses without reading a message. The values are in numbers of milliseconds.
requeue	String	None	The name of a destination, queue, or topic on which to requeue messages that Adaptive Server cannot process. If requeue is not specified, and the message cannot be processed, an error message appears.The endpoint specified must be on the same messaging provider as msgconsume and msgrecv.

Examples

Example 1 Receives a message from the specified *end_point*:

```
select msgrecv
    ('tibco_jms:tcp://my_jms_host:7222?queue=queue.sample,'
    +'user=jms_user1,password=jms_user1_password')
```

Example 2 Receives a message from the specified *end_point*, using the timeout option and specifying a message selector:

```
declare @mymsg varchar (16384)
select @mymsg = msgrecv('my_jms_provider?queue=queue.sample',
     OPTION 'timeout=1000'
     MESSAGE SELECTOR 'correlationID = ''MSG_001''')
```

Example 3 Forwards a message to the specified endpoint:

```
select msgsend(msgrecv('my_jms_provider?queue=queue.sample'),
     'another_jms_provider?queue=queue2')
```

Example 4 This msgrecv call only consumes messages from *queue.sample* when the message property "Name" is equal to "John Smith":

```
select msgrecv('my_jms_provider?queue=queue.sample',
     MESSAGE SELECTOR 'Name=''John Smith''')
```

Example 5 Illustrates how to insert a text message into a table:

```
create table T1(c1 numeric(5,0)identity, m text)
insert into T1
select msgrecv('my_jms_provider?queue=queue.sample',
     RETURNS text)
```

Example 6 This example reads a message and returns it as a varbinary.

```
select msgrecv('my_jms_provider?queue=queue.sample'
     returns varbinary(500))
```

Usage

- msgrecv receives a message from a specified *service_provider* and *service_definition*, and returns that message.
- By default, msgrecv is a blocking command, which blocks the message until it reads the next message from the message bus. If timeout is not -1, msgrecv returns null when the timeout interval lapses without reading a message. Its values are in number of milliseconds.
- Adaptive Server handles only messages of types message, text, or bytes. If Adaptive Server encounters a message it cannot process, and requeue is not specified, the message is left on the original queue. Subsequent reads encounter the same message, with the same effect. To prevent this behavior, specify requeue.When requeue is specified, messages that Adaptive Server cannot handle are placed on the queue specified.

  The endpoint specified must exist on the same messaging service provider as the endpoint used in msgrecv.
- The message includes the binary value of the datatype according to the byte ordering of the host machine.
- If you specify a filter parameter, the filter value is passed directly to the message provider. How it is used depends on the message provider.

- Comparisons specified in the message filter use the sort order specified by the message provider, which may not be the same used by Adaptive Server.
- JMS message providers use a JMS message selector as a filter. The rules for JMS message selectors are:
  - The syntax for the message selector is a subset of conditional expressions, including not, and, or, between, and like.
  - Identifiers are case sensitive.
  - Identifiers must designate message header fields and property names.
- Put apostrophes (') around option values to treat them as strings. If you omit the apostrophes , the option value is treated as another property name, and the expression is only true if the two properties have the same value.

  If your application uses quoted identifiers, the message selector must be enclosed in apostrophes (''). This means that if there are string values in your selectors, you must surround these values with double apostrophes (''). For example:

```
set quoted_identifier on
select msgrecv ('my_jms_provider?queue=queue.sample',
     MESSAGE SELECTOR 'color = ''red''')
```

  If your application does not use quoted identifiers, the message selector can be enclosed by ordinary double quotation marks. For example:

```
set quoted_identifier off
select msgrecv('my_jms_provider?queue=queue.sample',
     MESSAGE SELECTOR "color='red'")
```

  In this example, a messaging client application sends a message expressing a property named “color” to have the value “red”, and a property named “red” to have the value “color”.

```
select msgsend ('Sending message with property color',
     'my_jms_provider?queue=queue.sample'
     MESSAGE PROPERTY 'color=red, red=color')
```

  A client application that wants to consume only messages containing a property named “color” having the value “red” must place double apostrophes (''') around the selector value. For example:

```
select msgrecv('my_jms_provider?queue=queue.sample'
```

```
     MESSAGE SELECTOR 'color=''red''')
```

However, the message is not received if the client application uses the following syntax, because “red” is treated as a property name:

```
select msgrecv('my_jms_provider?queue=queue.sample',
     MESSAGE SELECTOR 'color=red')
```

In another example, a client sends a message that selects and filters for more than one property:

```
select msgsend('Sending message with properties',
     'my_jms_provider?queue=queue.sample',
     MESSAGE PROPERTY 'color=red, shape=square'
```

If another client wants to select messages in which property “color” equals “red” and property “shape” equals “square”, that client must execute the following:

```
select msgrecv('my_jms_provider?queue=queue.sample',
     MESSAGE SELECTOR 'color=''red'' and shape=''square''')
```

- Calling msgrecv has these results:
  - The value returned is the *message_body* value returned by the message provider, converted to the specified returns type.
  - The values of *@@msgheader* and *@@msgproperties* are set to those of <*msgheader*> and <*msgproperties*> documents, which contain the properties of the message returned by msgrecv.
  - You can extract the values of a specific property from a <*msgheader*> or <*msgproperties*> document with msgpropvalue. For details, see msgpropvalue on page 42.
  - The general format of <*msgheader*> and <*msgproperties*> is described in “Message-related global variables” on page 47.

msgsubscribe and msgunsubscribe

Description

Provides a SQL interface to subscribe or unsubscribe to a topic.

Syntax

msg_subscribe::= msgsubscribe (*subscription_name*)
msg_unsubscribe::=msgunsubscribe
(*subscription_name* [with {*remove* | *retain*}])
subscription_name::=*basic_character_expression*

Parameters

subscription_name
The name of the subscription to which you are subscribing. A *basic_character_expression*.

with{remove | retain}
Removes or retains the durable subscription from the JMS message provider.

Examples

Example 1 Tells the JMS messaging provider to begin holding messages published to the topic registered as “subscription_1”:

```
select msgsubscribe ('subscription_1')
```

Example 2 Tells the JMS messaging provider to stop holding messages published to the topic registered as “subscription_1”:

```
select msgunsubscribe('subscription_1')
```

Usage

- Before you specify a subscription with msgsubscribe or msgunscunscribe, you must register the subscription with sp_msgadmin. This example registers the durable subscription 'subscription_1':

```
sp_msgadmin 'register', 'subscription', 'subscription_1',
     'my_jms_provider?topic=topic.sample,user=user1,password=pwd',
     'Supplier=12345', null, 'durable1', 'client1'
```

- Once msgsubscribe is called, all messages published on the specified topic that qualify for the selector are held until msgconsume is called to read the messages. If you do not want to hold messages that arrive before you are ready to consume them, do not call msgsubscribe. Calling msgconsume without previously calling msgsubscribe starts the subscription when msgconsume is called.
- msgsubscribe starts a subscription for the client to receive messages defined by the endpoint and filter specified by *subscription_name*. It returns a 0 if it succeeds, or 1 if it fails.
- msgunsubscribe stops any current subscription for the client to the endpoint and filter specified by *subscription_name*. It returns a 0 if it succeeds, or 1 if it fails.
- If you specify with retain, the connection to the JMS messaging provider is terminated so that another subscription can connect, using the same subscriber *client_id* specified in the subscription. The durable subscriber remains defined within Adaptive Server and within the JMS message provider. If you specify with remove, the durable subscriber definition is removed from the JMS message provider. The default value is with retain.

When you unsubscribe a subscription using with remove, it is possible to miss messages:

```
<login>
select msgsubscribe('subscription_1')
select msgconsume('subscription_1')
...
select msgconsume('subscription_1')
select msgunsubscribe('subscription_1' WITH REMOVE)
<logout>

----Messages published to the topic registered as subscription_1 are no
----longer held by the JMS provider

<login>
select msgsubscribe('subscription_1')
select msgconsume('subscription_1')
...
select msgconsume('subscription_1')
select msgunsubscribe('subscription_1' WITH REMOVE)
```

In a separate scenario, a SQL session releases a subscription so that another session can consume messages. This example shows session 1 releasing the subscription, so that Session 2 can begin consuming from it.

Table 5-4: SQL sessions

Session 1	Session 2
`select msgunsubscribe ('subscription_1'WITH RETAIN)`	
`selectmsgconsume ('subscription_1')`	
`...`	
`selectmsgconsume ('subscription_1')`	
`select msgunsubscribe ('subscription_1' WITH RETAIN)`	
`...`	
	`select msgsubscribe('subscription_1')`
	`select msgconsume('subscription_1')`
	`...`
	`select msgconsume('subscription_1')`
	`select msgunsubscribe('subscription_1' WITH RETAIN)`

- The following example shows msgsubscribe used before the application logic is ready to read the messages that force the JMS client to hold messages. The application subscribes:

  ```
  select msgsubscribe ('subscription_1')
  ```

  The client consumes the message multiple times, and uses other application logic not related to messaging. Then it is ready to read messages, and it receives all the messages that have arrived since msgsubscribe was called:

  ```
  select msgconsume('subscription_1')
  select msgconsume('subscription_1')
  ```

  The client application is finished with this subscription, and unsubscribes:

  ```
  select msgunsubscribe('subscription_1')
  ```

msgpublish

Description

Provides a SQL interface to publish messages to topics.

Syntax

msgpublish_call::=msgpublish
(*message_body*, *subscription_name* [*options_and_properties*)
options_and_properties::=
[*option_clause*] [*properties_clause*]
option_clause::=[,] option *option_string*]
properties_clause::=[,]message property *option_string*
message_body::=
scalar_expression
| (*select_for_xml*)

Parameters

message_body

The message you are sending. The message body can contain any string of characters. It can be binary data, character data, or SQLX data.

subscription_name

Name of the subscription to which you are publishing messages.

option_string

The general format of an *option_string* is specified in the section option_string—general format.

The options you can specify for msgsend are in Table 5-1:

Table 5-5: Options for msgpublish

Option	Value	Default	Comments
schema	no yes “user_schema”	no	• *user_schema* is a user-supplied schema describing the *message_body*. • ‘no’ means that no schema is generated and sent out as part of the message. • ‘yes’ means that Adaptive Server generates an XML schema for the message. ‘yes’ is only meaningful in a *message_body* that uses the *select_for_xml* parameter. *select_for_xml* generates a SQLX-formatted representation of the SQL result set. The generated XML schema is a SQLX-formatted schema that describes the result set document. The schema is included in the message as ASE_MSGBODY_SCHEMA property.
type	text | bytes	text	The message type to send.

option_string

The general format of an *option_string* is specified in the section option_string—general format.

The options you can specify for msgpublish are in Table 5-5.

properties_clause

Is either an *option_string* or one of the options listed in Table 5-6. The options described in Table 5-6 are set as a property in the message header or message properties, as indicated in the disposition column of the table. The option value is the property value.

Property names are case sensitive.

If you use a property not listed in Table 5-2, it is set as a property in the message properties of the message sent.

Table 5-6: Option values for properties of msgpublish

Option	Values	Default	Disposition	Comments
mode	persistent or nonpersistent	persistent	header	If the mode is persistent, the message is backed by the JMS provider, using stable storage. If the messaging provider crashes before the message can be consumed, the message is lost, unless mode is set to persistent. If the mode is nonpersistent, and the messaging providercrashes, you may lose a message before it reaches the desired destination.
ttl	$0 – (2^{63}-1)$	0	header	ttl refers to time-to-live on the messaging bus. Adaptive Server is not affected by this. Expiry information, which is the duration of time during which the message is valid, in milliseconds. For instance, 60 means that the life of the message is 60 milliseconds. A value of 0 means that the message does not expire.
priority	1 to 10	4	header	The behavior of priority is controlled by the underlying message bus. The values mentioned here apply to TIBCO_JMS. Priorities from 0 to 4 are normal; priorities from 5 to 9 are expedited.
correlation	string	none	header	Client applications set correlation IDs to link messages together. Adaptive Server sets the correlation ID the application specifies.
replyqueue replytopic	A string containing a *queue_name* A string containing a *topic_name*	none none	header header	The value of *queue_name* or *topic_name* must be syb_temp. The type of the temporary destination, queue or topic, depends on whether you specify replyqueue or replytopic. Only the option listed last is used. Adaptive Server creates a temporary destination and sends information related to the newly created temporary destination as a part of the header information.

scalar_expression

If a message is a SQL *scalar_expression*, it can be of any datatype.

If the type option is not specified, the message type is text if the *scalar_expression* evaluates to a character datatype; otherwise the message type is bytes.

If the datatype of the *scalar_expression* is not character, it is converted to varbinary using the normal SQL rules for implicit conversion. The binary value of the datatype is included in the message according to the byte ordering of the host machine.

select_for_xml

A select expression that specifies a `for xml` clause.

Examples

Example 1 To publish messages, a subscription must be defined on the server to which the client is connected:

```
sp_msgadmin 'register', 'subscription','subscription_1',
     'my_jms_provider?topic=topic.sample,user=user1,password=pwd',
     'Supplier=12345',null, 'durable1', 'client'
```

Example 2 Then the client server can publish a message to a specified subscription:

```
select msgpublish
     ('Sending order', 'subscription_1',
     MESSAGE PROPERTY 'Supplier=12345')
```

Usage

- The *subscription_name* must have been specified in a call to:

  ```
  sp_msgadmin 'register', 'subscription'
  ```

  It should not be specified in a subsequent call to:

  ```
  sp_msgadmin 'remove', 'subscription'
  ```

msgconsume

Description

Provides a SQL interface to consume messages that are published to different topics.

Syntax

msgconsume_call ::=
  msgconsume (*subscription_name option _and_returns*)
option_and_returns ::=
  [*option_clause*] [*returns_clause*]
subscription_name:= *basic_character_expression*

option_clause::= [,] option *option_string*
returns_clause ::= [,] returns
SQL_type ::=
varchar(*integer*) | java.lang.String | *text*)
| varbinary(*integer*) | image

Parameters

subscription_name
Name of the subscription from which you are consuming messages.

option_string
The general format of the option_string is specified in option_string—general format. The special options to use when consuming a message are described in Table 5-7 on page 40.

returns
The clause that you want returned. If you do not specify a datatype to be returned, the default is varchar(16384). The legal SQL datatypes are:

- varchar(n)
- text
- java.lang.String
- varbinary(n)
- image
- univarchar(n)

basic_character_expression
A Transact-SQL query expression whose datatype is char, varchar, or java.lang.String.

option_string
is one of the following:

Table 5-7: msgconsume option values

Option	Values	Default	Values
timeout	-1, 0 – (2^{31}– 1)	-1	By default, msgconsume is a blocking command, which blocks the message until it reads the next message from the message bus. If timeout is not -1, msgconsume returns null when the timeout interval lapses without reading a message.The values are in number of milliseconds.

Option	Values	Default	Values
requeue	String	None	The name of a destination, queue, or topic on which to requeue messages that Adaptive Server cannot process. If requeue is not specified, and the message cannot be processed, an error message appears.The endpoint specified must be on the same messaging provideras msgconsume and msgrecv.

SQL_type
The datatypes used in SQL statements.

Examples

Example 1 Defines a subscription on the client server, before consuming a message:

```
sp_msgadmin 'register','subscription','subscription_1',
     'my_jms_provider?topic=topic.sample,user=user1,password=pwd',
     'Supplier=12345',null,'durable1','client1'
```

Example 2 Before consuming messages from a subscription, a client must first subscribe to the subscription:

```
select msgsubscribe('subscription_1')
declare @mymsg varchar(16384)
select @mymsg = msgconsume('subscription_1')
```

Example 3 Declares variables and receives a message from the specified subscription:

```
declare @mymsg varchar (16384)
select @mymsg = msgconsume('subscription_1',
     OPTION 'timeout=0')
```

Example 4 Forwards a message:

```
select msgsend
     (msgconsume('subscription_1'),'my_jms_provider?queue=queue.sample')
```

Example 5 This example reads a message and returns it as a varbinary:

```
select msgconsume('subscription_1' returns varbinary(500))
```

Usage

- msgconsume reads a message from the topic defined by the *end_point* and *message_filter* specified by the *subscription_name*. It returns null if there is a timeout or error, or returns the payload of the message it reads.

- Adaptive Server handles only messages of types message, text, or bytes. If Adaptive Server encounters a message it cannot process, and requeue is not specified, the message is left on the original queue. Subsequent reads encounter the same message, with the same effect. To prevent this behavior, specify requeue.When requeue is specified, messages that Adaptive Server cannot handle are placed on the queue specified.

  The endpoint specified must exist on the same messaging service provider as the endpoint used in msgconsume.
- Adaptive Server issues an error message if the messaging providerissues messages of of types other than message, text or bytes, and if requeue is not specified.
- Calling msgconsume has these results:
  - The value returned is the *message_body* value returned by the message provider, converted to the specified returns type.
  - The values of *@@msgheader* and *@@msgproperties* are set to <*msgheader*> and <*msgproperties*> documents, which contain the properties of the message that is returned by msgconsume.
  - The general format of <*msgheader*> and <*msgproperties*> documents are described in "<*msgheader*> and <*msgproperties*> documents. See “Message-related global variables” on page 47.
  - You can extract the values of a specific property from XML documents <*msgheader*> and <*msgproperties*> , and other related functions, with msgpropvalue. For more details see msgpropvalue, below.

msgpropvalue

Description

Extracts a specified property from a <*msgheader*> or <*msgproperties*> document. There are five msgpropvalue functions: msgproplist, msgpropcount, msgpropname, msgpropvalue, and msgproptype.

Syntax

msgproplist_call::=
msgproplist([msg_doc] [returns *varchar* | *text*]))
msgpropcount_call ::=
msgpropcount([*msg_doc*])
msgpropname_call ::=
msgpropname(*integer*[,*msg_doc*]),)

msgpropvalue_call::=
msgpropvalue(*prop_name* [, *msg_doc*])
msgproptype_call::=
msgproptype(*prop_name* [, *msg_doc*])
msg_doc ::= *basic_character_expression*
prop_name::= *basic_character_expression*

Parameters

msgproplist
Returns a string in the format of an *option_string* with all of the property attributes of msg_doc.

msgpropcount
Returns the number of properties, or attributes, in msg_doc.

msgpropname
Returns the name of msg_doc's Ith property, where I is the value of the integer parameter. The result is a null value if the value of the integer parameter is less than one or greater than the number of properties in msg_doc.

msgpropvalue
Returns the value for the msg_doc property whose name equals *prop_name*. The result is the property value converted to varchar, and is null if msg_doc does not have a property whose name is equal to *prop_name*.

msgproptype
Returns the message provider's property type for the msg_doc property whose name equals *prop_name*. The result is null if msg_doc does not have a property whose name is equal to *prop_name*.

msg_doc
The <*msgheader*> or <*msgproperties*> XML document. A *basic_character_expression*. If msg_doc is not specified, the current value of *@@msgprpoperties* is used.

prop_name
The property name from which you want to extract a value or type. A *basic_character_expression*.

Examples

Example 1 The following examples assume that a call from msgrecv returns a message whose single property is named trade_name and whose value is "Acme Maintenance" ("Quick & Safe"). The value of the *@@msgproperties* global variable is then:

```
<?xml version='1.0' encoding='UTF-8' standalone='yes' ?>
     <msgproperties trade_name='Acme Maintenance (&quot;Quick &amp;
     Safe&quot;)'></msgproperties>
```

The ampersand and the quotation marks surrounding the phrase `Quick & Safe` are replaced with the XML entities `"` and `&`,as required by XML convention.

Example 2 To retrieve the message property “trade name”:

```
select msgpropvalue(@@msgproperties, 'trade_name')
---------------
('Quick & Safe')Acme Maintenance
```

This is the original string that is stored in an Transact-SQL variable or column.

Example 3 To retrieve the list of properties belonging to a message, use one of the following examples:

```
select msgproplist()
```

```
select msgproplist(@@msgproperties)
```

Example 4 The following query to msgpropvalue returns null because the message retrieved does not have a property named “discount”:

```
select msgpropvalue('discount', @@msgproperties)
```

Example 5 To retrieve the number of properties from the last message retrieved:

```
select msgpropcount(@@msgproperties)
```

Example 6 To retrieve the 8th property from the last message retrieved:

```
select msgpropname(8, @@msgproperties)
```

Example 7 This query returns null, because the 9th property does not exist:

```
select msgpropname(9, @@msgproperties)
```

Example 8 To retrieve the value of the 8th property:

```
select msgpropvalue (msgpropname(8, @@msgproperties))
```

Usage

- If you omit the *msg_doc* parameter, the value of *@@msgproperties* is used.
- If the result of the msgproplist call is more than 16K, the result value contains the words “TRUNCATED”. You should specify “RETURNS text” instead, in this case. You must use other msgprop functions to iterate through the property list and get the names and values of the properties.

- If you run msgproplist without a return length, any output over the default return value (32) is truncated. To avoid this, specify the length of your returns. For example, this statement is truncated:

```
declare @properties varchar(1000)
select @properties = msgproplist(@@msgproperties returns varchar)
```

This one is not:

```
declare @properties varchar (1000)
select @properties= msgproplist(@@msgproperties returns varchar(1000))
```

CHAPTER 6

Common Topics in Real Time Messaging

This chapter discusses global variables, properties, and XML documents that are used by several of the functions and procedures in Chapter 5, “Real Time Data Functions,”, and in Chapter 4, “Real Time Stored Procedures for Administration.”

Message-related global variables

These global variables are used by several of the functions discussed in Chapter 5, “Real Time Data Functions.”

Description

Table 6-1 on page 47 lists global variables set by msgsend, msgrecv, msgpublish, and msgconsume.

Table 6-1: Messaging global variables

Variable name	Description	Functions that set the variable
@@msgheader	Contains message header information from the last message received. This variable’s format resembles XML. For details about this format, see “<msgheader> and <msgproperties> documents” on page 49.	msgrecv, msgconsume

Variable name	Description	Functions that set the variable
@@msgproperties	Contains message properties information from the message last received.	msgrecv, msgconsume
@@msgtimestamp	Contains the timestamp included in the message last sent.	msgsend, msgpublish
@@msgid	Contains the ID of the last message sent or received.	msgsend, msgpublish, msgrecv, msgconsume
@@msgstatus	Contains either the integer error code of the service provider exception, or zero, if the last operation did not raise an exception.	msgsend, msgpublish, msgrecv, msgconsume
@@msgstatusinfo	Contains either the error message of the service provider exception, or zero, if the last msgsend, msgpublish, msgrecv, or msgconsume raised an exception, or an empty string.	msgsend, msgpublish, msgrecv, msgconsume
@@msgreplytoinfo	Contains the name (*provider_url*, *queue_name*, *topic_name*, *user_name*) of the topic or queue name used to receive the next message. Can be a permanent or temporary destination.	msgsend, msgpublish
@@msgschema	Contains the schema of the message or null. Contains the value of the Adaptive Server property *ase_message_body_schema*. For more information, see the description of the schema option in msgsend and msgpublish.	msgsend, msgpublish

Usage

- These global variables are char datatypes, of length 16384.
- If they have trailing blanks, you can remove them using rtrim().
- *@@msgreplytoinfo* contains reply destination information from the message header. It is formatted as an end point, as described in msgsend on page 23:

  The password is not included in the value of this variable. If you want to use this destination as an argument in a subsequent msgsend or msgrecv call, add

  ```
  password=<your password>
  ```

<msgheader> and <msgproperties> documents

Description

The global variables *@@msgheader* and *@@msgproperties* are set with XML <*msgheader*> and <*msgproperties*> documents that contain the header and properties of the returned message. This section specifies the format of those documents.

The general format of a <*msgheader*> or <*msgproperties*> document for properties named PROPERTY_1, PROPERTY_2, and so on has the form described by the DTD templates in the following syntax section.

Syntax

```
<!DOCTYPE msgheader [
<!ELEMENT msgheader EMPTY>
<!ATTLIST property_1 CDATA>
<!ATTLIST property_2 CDATA>
etc.
<!DOCTYPE msgproperties [
<!ELEMENT msgproperties EMPTY>
<!ATTLIST property_1 CDATA>
<!ATTLIST property_2 CDATA>
```

Examples

These examples show <*msgheader*> or <*msgproperties*> documents for two select statements:

```
select msgsend('Sending message with properties',
               'my_jms_provider?queue=queue.sample',
               MESSAGE PROPERTY 'color=red, shape=square')

select msgrecv('my_jms_provider?queue=queue.sample')

select rtrim (@@msgproperties)

<?xml  version='1.0' encoding='UTF-8' standalone='yes' ?>
```

```
<msgproperties  RTMS_MSGBODY_FORMAT='&apos;string&apos;' ASE_RTMS_CHARSET='1'
ASE_RTMS_VERSION='&apos;1.0&apos;' ASE_VERSION='&apos;12.5.0.0&apos;'
shape='&apos;square&apos;' color='&apos;red&apos;' ></msgproperties>
```

```
select rtrim (@@msgheader)
```

```
<?xml  version='1.0' encoding='UTF-8' standalone='yes' ?>
<msgheader  type='&apos;NULL&apos;' timestamp='1080092021000'
replyto='&apos;queue.sample&apos;' redelivered='false' priority='4'
messageid='&apos;ID:E4JMS-SERVER.73018656B39:1&apos;' ttl='0'
destination='&apos;queue.sample&apos;' mode='2' correlation='&apos;NULL&apos;'
encoding='&apos;NULL&apos;' ></msgheader>
```

Usage

- A <*msheader*> or <*msgproperties*> document for a specified message contains one attribute for each property of the message header or each property of the message properties. The name of the attribute is the name of the property, and the value of the attribute is the string value of the value of the property.
- The values of attributes in <*msgheader*> or <*msgproperties*> documents are replaced with XML entities. msgpropvalue and msgpropname implicitly replace XML entities with attribute values.
- A <*msgheader*> or <*msgproperties*> document generated by msgrecv or msgconsume has an XML declaration that specifies the character set of the properties.

Adaptive Server-specific message properties

To help with debugging, monitoring, and so forth, predefined properties specific to Adaptive Server are included in the properties portion of the message. These properties typically handle messages that originate from another Adaptive Server, or are useful in debugging.

Many of these message properties are included only when running diagserver, or when certain trace flags are turned on. All properties beginning with “ASE_” are reserved. They cannot be set using msgsend or msgpublish.

Table 6-2 describes these message properties.

Table 6-2: Adaptive Server-specific messages

Property	Description	When to use
ASE_RTMS_CHARSET	Character set encoding of data sent.	Always
ASE_MSGBODY_SCHEMA	The schema describing the message body (payload), or null. This schema is nonnull only if the user sends the message schema as part of msgsend. If ASE_MSGBODY_FORMAT is xml, this property contains the XML schema describing the message body. This schema is not truncated, even if its value exceeds 16K.	Always
ASE_MSGBODY_FORMAT	The format of the message body: xml, string (in server character set), binary, and unicode (unichar in network order).	Always
ASE_ORIGIN	Name of the originating Adaptive Server.	Present with diagserver.
ASE_RTMS_VERSION	Version of Real Time Messaging Services in Adaptive Server.	Always
ASE_SPID	SPID that sent the message.	Present with diagserver.
ASE_TIMESTAMP	The timestamp of Adaptive Server showing the time the message was sent.	Present with diagserver.
ASE_VERSION	Version of Adaptive Server that published message.	Always
ASE_VERSIONSTRING	Version string of the Adaptive Server. Gives information about platform, build type, and so on. Useful for debugging.	Present with diagserver.

Note These properties are shown for informational purposes only. They may change in the future.

New keywords

Table 6-3 shows the keywords specific to Real Time Messaging Services, and the functions in which these keywords can be legally used.

Table 6-3: Double and triple keywords in RTMS

Keywords	Legal commands and functions using keywords
MESSAGE PROPERTY	select msgsend(,,, MESSAGE PROPERTY,,,) select MSGPUBLISH(,,,MESSAGE PROPERTY,,,)
MESSAGE SELECTOR	select msgrecv(,,,MESSAGE SELECTOR,,,) select msgconsume(,,,MESSAGE SELECTOR,,,)
WITH RETAIN	select msgunsubscribe(,,,WITH RETAIN,,,)
WITH REMOVE	select msgunsubscribe(,,,WITH REMOVE,,,)
TRANSACTIONAL MESSAGING NONE	set TRANSACTIONAL MESSAGING NONE
TRANSACTIONAL MESSAGING SIMPLE	set TRANSACTIONAL MESSAGING SIMPLE
TRANSACTIONAL MESSAGING FULL	set TRANSACTIONAL MESSAGING FULL

option_string—general format

Description

Specifies the general syntax and processing for *option_string*. Individual options are described in the functions that reference them.

Syntax

option_string::= *basic_character_expression*
option_string_value::= *option_and_value* [[,] *option_and_value*]
option_and_value ::= *option_name* = *option_value*
option_name::= *simple_identifier*
option_value::= *simple_identifier*
| *quoted_string*
| *integer_literal*
| *float_literal*
| *byte_literal*
| true
| false
| null

Parameters

option_string
String describing the option you want to specify.

simple_identifier
String that identifies the value of an *option*.

quoted_string
String formed using the normal SQL conventions for embedded quotation marks.

integer_literal
Literal specified by normal SQL conventions.

float_literal
Literal specified by normal SQL conventions.

true
A Boolean literal.

false
A Boolean literal.

null
A null literal.

byte_literal
Has the form 0xHH, where each H is a hexadecimal digit.

Usage

For *option_string* usage, see msgsend on page 23.

CHAPTER 7

Transactional Behavior

This chapter describes transactional message requirements and behavior.

Transactional message behavior

By default, all messaging operations, msgsend, msgrecv, msgpublish, msgconsume, msgsubscribe, and msgunsubscribe roll back if the database transaction rolls back. However, failure to accomplish a messaging operation with msgsend or msgrecv does not affect the parent database transaction.

- If a process included in a transaction executes msgsend or msgpublish, the resulting message is not visible on the message bus until the process commits the transaction. This is unlike executing a SQL update or insert.

  A process that executes SQL update and insert commands in a transaction sees the effect of these commands immediately, before they are committed.

- A process executing msgsend or msgpublish in a transaction to send a message cannot read that message using msgrecv or msgconsume until it commits the transaction.

Transactional messaging *set* option

Transactional behavior is controlled by the command set transactional messaging. This command provides three modes of operation, allowing you to select preferred behavior when you use messaging functions in a transaction:

```
set transactional messaging [ none | simple | full]
```

- *none* – provides that messaging operations and database operations do not affect each other. In this example, msgsend is executed and the message goes to the message bus, whether insert succeeds or fails:

```
begin tran
    msgsend (...)
    insert (...)
rollback
```

- *simple* (the default setting) – causes database operations to affect messaging operations, but messaging operations do not affect the database transaction. In this example, insert is not aborted if msgsend fails:

```
begin tran
    insert (...)
    msgsend (...)
commit
```

  In this example, msgsend is rolled back:

```
begin tran
    insert (...)
    msgsend (...)
rollback
```

- *full* – provides full transactional behavior. In this mode, messaging operations and database operations affect each other. If the messaging operation fails, the transaction rolls back. If database transactions fail, messaging operations roll back.

```
begin tran
    select @message=msgrecv(Q1,...)
    insert t2 values (@message,...)
    select msgsend ( t2.status,...)
commit tran
```

- When transactional messaging is set to *full* or *simple*, uncommitted transactions that send or publish messages cannot be read within the same transaction.

Transact-SQL applications can specify a preferred mode, depending on their application requirements.

Note You cannot use set transactional messaging inside a transaction.

CHAPTER 8

Samples

This chapter describes sample code that is distributed with Adaptive Server Real Time Messaging Services. This code illustrates messaging functionality.

Sybase directories

These code samples are in the directory:

$SYBASE/$SYBASE_ASE/samples/messaging

This directory contains three subdirectories:

Directory	Description
functionstring	Scripts to generate Replication Server function strings, for converting the default SQL template into calls to the messaging system
sql	SQL scripts with samples using Real Time Messaging
jdbc	JDBC samples using Real Time Messaging

Each subdirectory contains a README file, which explains the purpose of each code sample, provides a procedure for running it, and gives any installation instructions necessary.

The operating system file names in Windows and other platforms do not have exactly the same names. For example, *queue_listener.bat* on a Windows platform may be simply *queue_listener* on a UNIX/LINUX platform.

Using code samples with Replication Server function strings

These code samples assume some basic knowledge of Replication Server setup and configuration, and a basic knowledge of messaging.

The code samples in *$SYBASE/$SYBASE_ASE/samples/messaging/functionstring* are designed to help you use Adaptive Server RepAgent and Replication Server for publishing database modifications, such as the commands insert, update, and delete. They also demonstrate using stored procedures as a customized message to the messaging system.

You can publish database modifications as messages without altering your application code, using the methods illustrated in these code samples. These code samples publish messages from any existing Adaptive Server (version 12.5.2 and earlier) or any non-Adaptive Server database into the message bus.

Using code samples with SQL

The code samples in *$SYBASE/$SYBASE_ASE/samples/messaging/sql* illustrate how you can write or modify SQL (stored procedures, triggers, and so forth), to publish customized messages to the messaging system.

These samples also illustrate using SQL code to consume messages from the message bus, using Adaptive Server as both a participant in messaging and as an application using the message bus.

Using code samples with Java/JDBC

The code samples in *$SYBASE/$SYBASE_ASE/samples/messaging/jdbc* describe how you can write or modify Java code to publish customized messages to the messaging system.

These samples also illustrate Java code that consumes messages from the message bus, using Adaptive Server as both a participant in messaging and as an application using the message bus.

Index

Symbols

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

W

X