SYBASE®

Messaging Services User's Guide

Adaptive Server® Enterprise

15.0

DOCUMENT ID: DC20154-01-1500-01

LAST REVISED: September 2005

Contents

About This Book

Audience

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

How to use this book

This book assists you in configuring and using real-time messaging in Adaptive Server database applications. It includes these chapters:

- Chapter 1, “Introduction,” discusses messaging concepts, models, and formats, and provides a short glossary of terms.
- Chapter 2, “Understanding Real-Time Data Services,” is an overview of Real-Time Data Services (RTDS) specific to Adaptive Server.
- Chapter 3, “Configuring Real-Time Data Services,” provides a procedure for configuring your system.
- Chapter 4, “SQL Reference,” documents the SQL stored procedures, functions, and global variables for managing and administering real-time messaging, and the general format of option strings.
- Chapter 5, “Transactional Behavior,” describes transactional message requirements and behavior.
- Chapter 6, “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 Message Service message bus by TIBCO Software at http://www.tibco.com.

Related documents

The Sybase® Adaptive Server Enterprise documentation set consists of the following:

- 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 15.0, the system changes added to support those features, and changes that may affect your existing applications.
- *ASE Replicator User's Guide* – describes how to use the Adaptive Server 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.
- The *Configuration Guide* for your platform – provides instructions for performing specific configuration tasks for Adaptive Server.
- *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.
- *Java in Adaptive Server Enterprise* – describes how to install and use Java classes as data types, functions, and stored procedures in the Adaptive Server database.
- *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).
- *Messaging Service User's Guide* – describes how to useReal Time Messaging Services to integrate TIBCO Java Message Service and IBM WebSphere MQ messaging services with all Adaptive Server database applications.

- *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. Full size 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.
- *Unified Agent and Agent Management Console* – Describes the Unified Agent, which provides runtime services to manage, monitor and control distributed Sybase resources.
- *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 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://www.sybase.com/support/manuals/.

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.

❖ **Finding the latest information on component certifications**

1. Point your Web browser to Availability and Certification Reports at http://certification.sybase.com/.
2. Either select the product family and product under Search by Product; or select the platform and product under Search by Platform.
3. Select Search to display the availability and certification report for the selection.

❖ **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 EBFs/Maintenance releases is displayed.

   Padlock icons indicate that you do not have download authorization for certain EBFs/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 EBFs/Maintenance report, or click the product description to download the software.

Conventions

The following sections describe conventions used in this manual.

SQL is a free-form language. There are no rules about the number of words you can put on a line or where you must break a line. However, for readability, all examples and most syntax statements in this manual are formatted so that each clause of a statement begins on a new line. Clauses that have more than one part extend to additional lines, which are indented. Complex commands are formatted using modified Backus Naur Form (BNF) notation.

Table 1 shows the conventions for syntax statements that appear in this manual:

Table 1: Font and syntax conventions for this manual

Element	Example
Command names,procedure names, utility names, and other keywords display in sans serif font.	select sp_configure
Database names and datatypes are in sans serif font.	master database

Element	Example
Book names, file names, variables, and path names are in italics.	*System Administration Guide* *sql.ini* file *column_name* *$SYBASE/ASE* directory

Variables—or words that stand for values that you fill in—when they are part of a query or statement, are in italics in Courier font.	`select` *`column_name`*
	    `from` *`table_name`*
	    `where` *`search_conditions`*

Type parentheses as part of the command.	`compute` *`row_aggregate`* `(`*`column_name`*`)`
Double colon, equals sign indicates that the syntax is written in BNF notation. Do not type this symbol. Indicates “is defined as”.	`::=`
Curly braces mean that you must choose at least one of the enclosed options. Do not type the braces.	`{cash, check, credit}`
Brackets mean that to choose one or more of the enclosed options is optional. Do not type the brackets.	`[cash | check | credit]`
The comma means you may choose as many of the options shown as you want. Separate your choices with commas as part of the command.	`cash, check, credit`
The pipe or vertical bar(|) means you may select only one of the options shown.	`cash | check | credit`

An ellipsis (...) means that you can *repeat* the last unit as many times as you like.	`buy thing = price [cash | check | credit]`
	`[, thing = price [cash | check | credit]]...`
	You must buy at least one thing and give its price. You may choose a method of payment: one of the items enclosed in square brackets. You may also choose to buy additional things: as many of them as you like. For each thing you buy, give its name, its price, and (optionally) a method of payment.

- Syntax statements (displaying the syntax and all options for a command) appear as follows:

  sp_dropdevice [*device_name*]

  For a command with more options:

  select *column_name*
     from *table_name*
     where *search_conditions*

  In syntax statements, keywords (commands) are in normal font and identifiers are in lowercase. Italic font shows user-supplied words.

- Examples showing the use of Transact-SQL commands are printed like this:

```
select * from publishers
```

- Examples of output from the computer appear as follows:

```
pub_id    pub_name                 city          state
-------   --------------------     -----------   -----
0736      New Age Books            Boston        MA
0877      Binnet & Hardley         Washington    DC
1389      Algodata Infosystems     Berkeley      CA

(3 rows affected)
```

In this manual, most of the examples are in lowercase. However, you can disregard case when typing Transact-SQL keywords. For example, SELECT, Select, and select are the same.

Adaptive Server's sensitivity to the case of database objects, such as table names, depends on the sort order installed on Adaptive Server. You can change case sensitivity for single-byte character sets by reconfiguring the Adaptive Server sort order. For more information, see the *System Administration Guide*.

Accessibility features

This document is available in an HTML version that is specialized for accessibility. You can navigate the HTML with an adaptive technology such as a screen reader, or view it with a screen enlarger.

Adaptive Server HTML documentation has been tested for compliance with U.S. government Section 508 Accessibility requirements. Documents that comply with Section 508 generally also meet non-U.S. accessibility guidelines, such as the World Wide Web Consortium (W3C) guidelines for Web sites.

Note You might 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 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.

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 that you have 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 aspects of messaging that are specific to Adaptive Server. This functionality referred to in this document as Real-Time Data Services (RTDS).

Adaptive Server 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 Data Services (RTDS), simplifies the development of such applications, using Adaptive Server with TIBCO Enterprise Messaging Service (EMS).

Note EMS is the TIBCO implementation of a Java messaging service (JMS). Since Adaptive Server only supports EMS, the reference to EMS and JMS refer to the same thing in this document.

Messaging systems allow senders and receivers to be detached. Not all components must be 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 running.

JMS are APIs that define 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 provider can implement architecture that is centralized or decentralized, or a hybrid of the two.

Adaptive Server performs messaging operations within SQL statements, using built-in functions.

Real-Time Data Services provide a way to capture transactions (data changes) in an Adaptive Server database and deliver them as events to external applications using JMS message bus, provided by TIBCO EMS.

Messaging models

This section describes the messaging models for JMS.

JMS defines two messaging models:

- Publish-and-subscribe (topics)
- Point-to-point (queues)

Publish-and-subscribe (topics)

The publish-and-subscribe model is a one-to-many model. In this type of messaging model, 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, rather than Adaptive Server, retains the message.

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

Point-to-point (queues)

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 provider ensures that the message is delivered to only one message receiver.

Message format

The message format for JMS consists of:

- Message header – contains fixed-size portions and variable-sized portions of information specified by the standard. Most of this information is automatically assigned by the message provider.
- Message body – is the application data that client applications exchange.

  JMSdefines structured message types, such as stream and map, and unstructured message types, such as text, byte, and object.

JMS message properties

Message properties are user-defined additional properties 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 for TIBCO EMS provide a way for message consumers to filter the message stream 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.

CHAPTER 2

Understanding Real-Time Data Services

This chapter provides an overview of Real-Time Data Services, which allow you to use Adaptive Server as a client of the message provider. You can send messages to or retrieve messages from the messaging provider by using Transact-SQL commands.

Sending and receiving messages from a queue

Using the built-in functions msgsend and msgrecv, Transact-SQL applications can send messages to a queue or read messages from a 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 (header or user properties) from relational data or from application logic, and include the constructed message properties in the message that you are sending.

Messages read from the message bus 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 49.

Publishing and consuming messages from a JMS 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 functions can reference. You can register a subscription as a **durable** or nondurable, 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. Use msgunsubscribe to delete durable subscriptions from the JMS provider.

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

See Chapter 4, “SQL Reference” for syntax, parameter, and usage information for sp_msgadmin and functions.

Working with message properties

When a message is read, the message header and user 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

These examples provide a brief preview of the Transact-SQL messaging interface.

Examples

Example 1 Sends a message to a queue:

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

Example 2 Reads a message from a queue, with and without a filter:

```
select msgrecv('tibco_jms:tcp://my_ems_host:7222?queue=queue.sample')

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

Example 3 Publishes a message to a topic:

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

Example 4 Consumes a message from a topic:

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

Example 5 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 Data Services

This chapter has instructions for installing and configuring Real-Time Data Services (RTDS) in Adaptive Server Enterprise.

Configuring RTDS

To install RTDS feature, install Adaptive Server as you normally would, and entering your ASE_MESSAGING or SY_RTDS license keys. Follow the instructions in the installation guide for your platform. You must install Adaptive Server before you can configure RTDS.

❖ Configuring RTDS

1 Install the RTDS 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 RTDS:

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

Note LD_LIBRARY_PATH must be correct for this step to succeed. See Table 3-1 on page 10 for information on how to set the shared library environment variables.

4 Increase memory configuration:

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

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

5 Add the local server, if you have not already, then restart:

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

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

6 Install the TIBCO EMS client libraries in your preferred location and define that location as *TIBEMS_ROOT*.

7 Set the shared library search environment variable before you restart Adaptive Server. Table 3-1 lists the variables and the library names.

Table 3-1: Shared library environment variables

Platform	**Directory**	**Library name**
AIX 64	*LIBPATH*	*$TIBEMS_ROOT/clients/c/lib/64:$TIBEMS_ROOT/clients/c/lib*
HP 64	*SHLIB_PATH* or *LD_LIBRARY_PATH*	*$TIBEMS_ROOT/clients/c/lib/64:$TIBEMS_ROOT/clients/c/lib*
Linux 32	*LD_LIBRARY_PATH*	*$TIBEMS_ROOT/clients/c/lib*
Solaris 32	*LD_LIBRARY_PATH*	*$TIBEMS_ROOT/clients/c/lib*
Solaris 64	*LD_LIBRARY_PATH*	*$TIBEMS_ROOT/clients/c/lib/64:$TIBEMS_ROOT/clients/c/lib*
Windows 2000	*PATH*	*%TIBEMS_ROOT%\clients\c\bin*

Table 3-2 shows the names of the client libraries.

Table 3-2: Client libraries

Platform	Library name
AIX 64	*libtibems64.so*
HP 64	*libtibems64.sl*
Linux 32	*libtibems.so*
Solaris 32	*libtibems.so*
Solaris 64	*libtibems64.so*
Windows 2000	*TIBEMS.DLL*

Configuring Adaptive Server to communicate with TIBCO Rendezvous Server

If you are running TIBCO Rendezvous Server and you want it to communicate with Adaptive Server, you must configure a connection bridge between the two so that Adaptive Server can read messages from and write messages to Rendezvous Server.

❖ Creating a connection bridge between Adaptive Server and TIBCO Rendezvous Server

1 In the *tibems.conf* file, enable the tibrv_transports parameter:

```
tibrv_transports = enabled
```

2 Add the transport in *transports.conf* file, where:

- RV – is the name of the RV transport.
- type – is the type of external messaging system. Options are tibrv or tibrvcm.
- service – is the RV service port. The default is 7500.
- network – is the subnet for the host.
- daemon – is the default daemon for Rendezvous Server.

Note You must include the path to Rendezvous Server in the *PATH* variable before you restart tibjmsd, the server processes that run the TIBCO messaging server.

In this example, the RV transport called RV1 is an external messaging system type tibrv that uses service port 7500 on subnet 10.22.102.0. Its daemon *tcp:bigcrunch:7223*, and its entry in the *transports.conf* file is:

```
[RV1]
type = tibrv
service = 7500
network = 10.22.102.0
daemon = tcp:bigcrunch:7223
```

3. Restart tibjmsd.

4. Add an import or export property to the create topic command. In this example, messages published to *topic.rv1* are automatically sent to the RV subject named *topic.rv1*, and messages sent to through RV to the RV subject named *topic.rv1* can be read from topic *topic.rv1*:

   ```
   create topic topic.rv1 import=rv1,export=rv1
   ```

5. Add the import property to create queue command. In this example, messages sent to through RV to the RV subject named *queue.rv1* can be read from queue *queue.rv1*:

   ```
   create queue queue.rv1 import=rv1
   ```

CHAPTER 4 SQL Reference

This chapter describes a stored procedure, sp_msgadmin and its options, which you can use to manage and administer Adaptive Server Messaging Services.

Messaging-related global variables

These global variables provide application programs with access to message information from the most recent message sent or received. They are discussed in Chapter 4, “SQL Reference.”

@@msgcorrelation

Contains correlation from last message sent or read.

@@msgcorrelation contains the correlationId from the the most recent message sent or received.

@@msgheader

Contains message header information from the most recent message received. This variable's format is in XML. For details about this format, see "<msgheader> and <msgproperties> documents" on page 15.

Functions that set *@@msgheader* include msgrecv and msgconsume.

Table 4-1: @@msgheader fields and descriptions

Property name	Description
correlation	Correlation ID from the message
destination	The name of the destination from the message
encoding	The encoding name of the message
messageid	The message ID from the message
mode	Delivery mode of the message. Values: • persistent • nonpersistent
priority	The message priority
redelivered	The redelivery status from the message
replyto	The replyto name from the message
timestamp	The message timestamp
ttl	The expiry from the message that indicates how long a message exists
type	The message type

@@msgid

Contains the ID of the most recent message sent or received.

Functions that set the variable are: msgsend, msgpublish, msgrecv, msgconsume.

@@msgproperties

Contains message properties information from the most recent message received. This variable's format is in XML. For details about this format, see "<msgheader> and <msgproperties> documents" on page 15.

The *@@msgproperties* are the user properties from the message.

Functions that set the variable are: 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.

Functions that set the variable is: msgsend

@@msgschema	Contains the schema of the message or a null value. 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. Functions that set the variable are: msgsend, msgpublish.
@@msgstatus	Contains either the integer error code of the service provider exception, or zero, if the last operation did not raise an exception. Functions that set the variable are: 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. Functions that set the variable are: msgsend, msgrecv.
@@msgtimestamp	Contains the timestamp included in the message last sent. Functions that set the variable are: msgsend, msgpublish.

Usage

- These global variables are char datatypes, of length 16384.
- You can remove trailing blanks using rtrim.
- *@@msgreplytoinfo* contains reply destination information from the message header. It is formatted as an endpoint, as described in msgsend on page 49.

  The password is not included in the value of *@@msgreplytoinfo*. 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>* and *<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_ems_provider?queue=queue.sample',
                message property 'color=red, shape=square')

select msgrecv('my_ems_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:E4EMS-SERVER.73018656B39:1&apos;'
    ttl='0'
    destination='&apos;queue.sample&apos;'
    mode='2'
    correlation='&apos;null&apos;'
    encoding='&apos;null&apos;' >
```

```
</msgheader>
```

Usage

- A *<msgheader>* or *<msgproperties>* document for a specified message contains one attribute for each property of the message header or 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 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 TIBCO EMS message. These properties typically handle messages that either originate from another Adaptive Server, or that may be useful in debugging.

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

Table 4-2 describes these message properties.

Table 4-2: Adaptive Server-specific messages

Property	Description	When to use
ASE_RTMS_CHARSET	Character set encoding of sent data.	Always
ASE_MSGBODY_SCHEMA	The schema describing the message body or a null value. This schema is non-null 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 payload. 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

Property	**Description**	**When to use**
ASE_ORIGIN	Name of the originating Adaptive Server.	Present with diagserver
ASE_RTMS_VERSION	Version of Adaptive Server using Real-Time Data Services.	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.

Keywords

Table 4-3 shows the keywords specific to Adaptive Server Messaging Services, and the functions in which these keywords can be legally used.

Table 4-3: Double and triple keywords in Adaptive Server Messaging Services

Keywords	Legal commands and functions using keywords
message header	select msgsend(,,, message header,,,) select msgpublish(,,,message header,,,)
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

Stored procedures

The stored procedures you use with this feature is sp_msgadmin on page 21.

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. See *Reference Manual: Procedures* for more information on how to use sp_msgadmin.

Built-in functions

The section in this chapter on built-in functions describes the SQL functions for administering Real-Time Data Services, and the general format of option strings. See Table 4-2 on page 17 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

The functions listed in this chapter, and their page numbers, are:

- msgconsume on page 28
- msgpropcount on page 31
- msgproplist on page 32
- msgpropname on page 34
- msgproptype on page 35
- msgpropvalue on page 37
- msgpublish on page 39
- msgrecv on page 43
- msgsend on page 49
- msgsubscribe on page 56

See *Reference Manual: Commands* for more information on how to use built-in functions in the where clause of a select statement.

Syntax segment

The section in this chapter on the option_string syntax segments describes the portions of SQL syntax and constraints used in administering Real-Time Data Services. The syntax segment listed in this chapter, and its page number is option_string on page 62.

sp_msgadmin

Description

Configures and administers messaging-related information.

Syntax

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

sp_msgadmin 'help'[, 'list' | 'register' | 'default' | 'remove']

sp_msgadmin "list" [, 'login' [, *provider_name* [, *login_name*]
        'provider' [, *provider_name*]

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

sp_msgadmin 'remove',
        [, '*provider*', *provider_name*
        | 'login', *provider_name*, *local_login* [, *role*]
        | 'subscription', *subscription_name*

Parameters

client_id

is the identification used by the messaging provider to identify the subscription as durable. *client_id* is a character string value. If you specify either *client_id* or *durable_name*, you must also specify the other, 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 cannot 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.

default
: specifies a default. In the case of sp_msgadmin 'list', lists the syntax to specify the default login for a specified message provider.

delivery_option
: species whether a SQL session can consume messages that it publishes. The valid values are:

- local – the SQL session can consume messages that it publishes.
- nonlocal – the SQL session cannot consume messages that it publishes.
- null – assumes the value is local.

durable_name
: is a character string value. See the description of *client_id*.

endpoint
: is the topic to which the subscription is addressed. See the description of *endpoint* in msgsend on page 49.

help
: provides syntax information about this stored procedure or about particular parameters.

list
: lists syntax information about message providers, logins, or subscriptions.

local_login
: is an Adaptive Server login that maps to the local login.

login
: lists information about a particular messaging provider login mapping or about all messaging provider logins. When used with:

- register – registers a login mapping.
- default – specifies a default login.
- remove – 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,...
```

login_name
: is a login name.

messaging_provider_URL
is the URL of the messaging provider you are registering.

provider
specifies the message provider. When used with:

- register – registers a message provider.
- list – lists information about a particular messaging provider or about all message providers.
- remove – removes a messaging provider previously defined by this call:

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

provider_class
is the class of the messaging provider you are adding. Valid value is “tibco_jms”.

provider_login
is the login name of the messaging provider that *local_login* maps to when connecting to the message provider. It is also the login the provider uses as the default login when sending or receiving messages from the messaging provider specified by *provider_name* when using sp_msgadmin 'default'.

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

provider_password
is the messaging provider password of the *provider_login*.

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

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

role_name
is 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*.

selector
is a message filter that allows a client to select messages of interest. See the description of filters in msgrecv on page 43.

subscription
lists information about a particular subscription or about all subscriptions. Specifies the message provider. When used with:

- register – registers a subscription.
- list – lists information about a particular subscription or about all subscriptions.
- remove – removes a subscription previously created by:

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

subscription_name
is a subscription name.

Examples

Example 1 Specifies the default login that applies to all unmapped Adaptive Server logins, when using a specified messaging provider for either sending or receiving:

```
sp_msgadmin 'default', 'login', 'my_wms_provider',
      'ems_user1', 'ems_user1_password'
```

Note You must first register the *provider_name* by calling sp_msgadmin 'register', 'provider'.

Example 2 Specifies the default login:

```
sp_msgadmin 'default', 'login', 'one_ems_provider',
     'loginsa', 'abcdef123456'
```

Example 3 Describes the syntax for sp_msgadmin 'list':

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

Example 4 Checks the default login:

```
sp_msgadmin 'list', 'login', 'my_ems_provider'
```

Example 5 Lists the details for the user with a login of “loginsa”:

```
sp_msgadmin 'list', 'login', 'my_ems_provider',
    'loginsa'
```

Example 6 Lists the details for the “my_ems_provider” message provider:

```
sp_msgadmin 'list', 'provider', 'my_ems_provider'
```

Example 7 Lists the details for subscription “subscription_1”:

```
sp_msgadmin 'list', 'subscription',
    'subscription_1'
```

Example 8 Registers the login “ase_login1”, using messaging provider login “ems_user1”, and messaging provider name “my_ems_provider”:

```
sp_msgadmin 'register', 'login', 'my_ems_provider',
    'ase_login1', 'ems_user1', 'ems_user1_password'
```

Example 9 Registers a login using the messaging provider login “ems_user1”, and a specified password used for all unmapped Adaptive Server logins:

```
sp_msgadmin 'register', 'login', 'my_ems_provider',
    null, 'ems_user1', 'ems_user1_password'
```

Example 10 Registers a login with the messaging provider login “ems_user1”, and a specified password used for all Adaptive Server logins that have sa_role permissions:

```
sp_msgadmin 'register', 'login', 'my_ems_provider',
    null, 'ems_user1', 'ems_user1_password',
    'sa_role'
```

Example 11 Registers the “my_ems_provider” messaging provider, which has a class of “tibco_jms” and an IP of 10.23.233.32:4823 as its address:

```
sp_msgadmin 'register', 'provider',
    'my_ems_provider', 'tibco_jms',
    'tcp://10.23.233.32:4823'
```

Example 12 Registers a durable subscription named “durable_sub1”, then sp_msgadmin 'list' displays information about the new subscription.

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

Example 13 Registers “subscription_1”, a nondurable subscription.

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

Note You must first use sp_msgadmin register, provider to register "my_ems_provider".

Example 14 Removes the default login:

```
sp_msgadmin 'remove', 'login', 'my_ems_provider'
```

Example 15 Removes the messaging provider "my_ems_provider":

```
sp_msgadmin 'remove', 'provider', 'my_ems_provider'
```

Example 16 Removes the Adaptive Server login "ase_login1" associated with the messaging provider "my_ems_provider":

```
sp_msgadmin 'remove', 'login', 'my_ems_provider',
    'ase_login1'
```

Example 17 Removes the default login, indicated by a null login parameter:

```
sp_msgadmin 'remove', 'login', 'my_ems_provider',
    null
```

Example 18 Removes all logins for role sa_role on "my_ems_provider":

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

Example 19 Removes "subscription_1":

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

Usage

You cannot use sp_msgadmin inside a transaction.

sp_msgadmin 'register'

- 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 endpoint, 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 21 for usage common to the variants of sp_msgadmin.

sp_msgadmin 'remove'

- 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.
- You must specify *local_login* as null if you specify *role_name*.

Permissions

You must have messaging_role to run the msgsend and msgrecv functions.

You must have messaging_role and sso_role permissions to issue:

- sp_msgadmin 'default'
- sp_msgadmim 'register'
- sp_msgadmin 'remove'

Any user can issue:

- sp_msgadmim 'help'
- sp_msgadmin 'list'

msgconsume

Description

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

Syntax

```
msgconsume_call ::=
        msgconsume (subscription_name, option_and_returns)
subscription_name:= basic_character_expression
option_and_returns ::= [option_clause] [returns_clause]
option_clause::= [,] option option_string
returns_clause ::= [,] returns sql_type
subscriber_name ::= basic_character_expression
SQL_type ::=
        varchar(integer) | java.lang.String | text)
        | varbinary(integer) |  image
```

Parameters

basic_character_expression

is a Transact-SQL query expression with datatype is char, varchar, or java.lang.String.

option_string

is the general format of *option_string* is specified in option_string on page 62. The special options to use when consuming a message are described in Table 4-4:

Table 4-4: option and option_string values for msgconsume

***option* values**	***option_string* values**	**Default**	**Description**
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 a null value when the timeout interval lapses without reading a message.The values are in number of milliseconds.
requeue	*string*	None	The name of a destination, queue, or topic on which to requeue messages that Adaptive Server cannot process. If you do not specify requeue, and the message cannot be processed, an error message appears.The endpoint specified must be on the same messaging provider as msgconsume and msgrecv.

subscription_name

is the name of the subscription from which you are consuming messages.

returns

specifies the clause that you want returned.

SQL_type
is the datatype used in SQL statements.

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

Examples

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

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

Before consuming messages from a subscription, the client first subscribes to the subscription:

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

Example 2 Declares variables and receives a message from the specified subscription:

```
declare @mymsg varchar (16384)
select @mymsg = msgconsume('subscription_1',
     option 'timeout=2000')
```

Forwards a message:

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

Reads a message and returns it as a varbinary:

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

Usage

- Unrecognized option names result in an error.
- msgconsume reads a message from the topic defined by the *end_point* and *message_filter* specified by the *subscription_name*. It returns a null value if there is a timeout or error, or returns the body 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 specified endpoint must exist on the same message service provider as the endpoint used in msgconsume.
- Adaptive Server issues an error message if the messaging provider issues messages 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 "Messaging-related global variables" on page 13.
  - 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.

Permissions

You must have messaging_role to run msgconsume.

msgpropcount

Description

Extracts and returns the number of properties or attributes in msg_doc from a <*msgheader*> and <*msgproperties*> document.

Syntax

msgpropcount_call ::= msgpropcount([*msg_doc*])
msg_doc ::= *basic_character_expression*
prop_name::= *basic_character_expression*

Parameters

msgpropcount_call
makes the request to use the msgpropcount function.

msg_doc
is the <*msgheader*> or <*msgproperties*> XML document in the form of *basic_character_expression*. If you do not specify *msg_doc*, msgpropcount uses the current value of @@*msgprpoperties*.

prop_name
is the property name from which you want to extract a value or type in the form of *basic_character_expression*.

Examples

This example assumes that a call from msgrecv returns a message with a single property named trade_name and value of “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 `&quot;` and `&amp;`, as required by XML convention.

Retrieves the number of properties from the last message retrieved:

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

msgproplist

Description

Extracts and returns from a <*msgheader*> and <*msgproperties*> document a string in the format of an *option_string* with all of the property attributes of msg_doc.

Syntax

msgproplist_call::= msgproplist([*msg_doc*] [returns *varchar* | *text*]))
    msg_doc ::= *basic_character_expression*
    prop_name::= *basic_character_expression*

Parameters

msgproplist_call
makes the request to use the msgproplist function.

msg_doc
is 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
is the property name from which you want to extract a value or type. A *basic_character_expression*.

returns *varchar* | *text*
specifies the format of the returning message.

Examples

This example assumes that a call from msgrecv returns a message with a single property named “trade_name” and value of “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 `&quot;` and `&amp;`, as required by XML convention.

Either of these retrieves the list of properties belonging to a message:

```
select msgproplist
```

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

Usage

- If the result of the msgproplist call is more than 16K, the result value contains the word “TRUNCATED”. You should specify “RETURNS text” instead, in this case. You must use other msgprop functions to iterate through the property list and obtain 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)
```

However, this one is not:

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

msgpropname

Description

Extracts and returns the property name from a <*msgheader*> and <*msgproperties*> document. 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.

Syntax

msgpropname_call ::= msgpropname(integer[,*msg_doc*]),)
msg_doc ::= *basic_character_expression*
prop_name::= *basic_character_expression*

Parameters

msgpropname_call
makes the request to use the msgpropname function.

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 This example assumes that a call from msgrecv returns a message with a single property named trade_name and value of “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 `&quot;` and `&amp;`, as required by XML convention.

Retrieves the eighth property from the most recent message retrieved:

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

Example 2 Returns a null value, because the ninth property does not exist:

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

msgproptype

Description

Extracts and returns from a <*msgheader*> and <*msgproperties*> document the message provider's property type for the msg_doc property with a name that equals *prop_name*. The result is a null value if msg_doc does not have a property with a name is equal to *prop_name*.

Syntax

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

Parameters

msgproptype_call
makes the request to use the msgproptype function.

msg_doc
is 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
is the property name from which you want to extract a value or type. A *basic_character_expression*.

Examples

A message is sent with two properties, "integer_prop," which is an integer with value 1234, and "string_prop," a string with the value "cat":

```
select msgsend('msgproptype example',
    'tibco_jms:tcp://localhost:7222?queue=queue.sample',
    MESSAGE PROPERTY "integer_prop=1234,string_prop='cat'")
go

---------------------------------------
ID:E4JMS-SERVER.82CC311EC:1
(1 row affected)
```

The message is then read back:

```
select msgrecv('tibco_jms:tcp://localhost:7222?queue=queue.sample')
go

---------------------------------------
msgproptype example

(1 row affected)
```

The *@@msgproperties* global variable is selected to display what the properties were on in the message just received:

```
select @@msgproperties
go
```

```
---------------------------------------
<?xml  version="1.0" encoding="UTF-8" standalone="yes" ?>
    <msgproperties
       string_prop="&apos;cat&apos;"
       ASE_RTMS_CHARSET="1"
       ASE_ORIGIN="&apos;francis_pinot_2&apos;"
       ASE_SPID="15"
       ASE_MSGBODY_FORMAT="&apos;string&apos;"
       ASE_TIMESTAMP="&apos;2005/06/22 15:01:36.91&apos;"
       ASE_MSGBODY_SCHEMA="&apos;NULL&apos;"
       ASE_RTMS_VERSION="&apos;1.0&apos;"
       ASE_VERSION="&apos;12.5.0.0&apos;"
       integer_prop="1234">
    </msgproperties>

(1 row affected)
```

The first msgproptype call asks for the type of the “integer_prop” property, and returns “Integer”:

```
1> select msgproptype('integer_prop')
2> go
---------------------------------------
Integer

(1 row affected)
```

The second msgproptype call asks for the type of the “string_prop” property, and returns “String”:

```
1> select msgproptype('string_prop')
2> go
---------------------------------------
String

(1 row affected)
```

msgpropvalue

Description

Extracts and returns from a <*msgheader*> and <*msgproperties*> document the value for the msg_doc property where the name equals *prop_name*. The result is the property value converted to varchar, and is a null value if msg_doc does not have a property with name that is equal to *prop_name*.

Syntax

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

Parameters

msgpropvalue_call
  makes the request to use the msgpropvalue function.

msg_doc
  is 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
  is the property name from which you want to extract a value or type. A *basic_character_expression*.

Examples

Example 1 These examples assume that a call from msgrecv returns a message with a single property named “trade_name” and value of “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 `&quot;` and `&amp;`, as required by XML convention.

Retrieves the message property trade_name:

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

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

Example 2 Returns a null value because the message retrieved does not have a property named “discount”:

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

Example 3 Retrieves the value of the eighth property:

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

msgpublish

Description

Provides a SQL interface to publish messages to topics.

Syntax

```
message_publish_call ::=
            msgpublish(message_body, subscription_name
                  [options_and_properties])
options_and_properties ::= [option_clause] [properties_clause]
                  [header_clause]
option_clause ::= [,] option option_string
header_clause ::= [,] message header option_string
properties_clause ::= [,] message property option_string
message_body ::= scalar_expression | (select_for_xml)
```

Parameters

message_body

is 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

is the name of the subscription to which you are publishing messages.

option_clause

is the general format of the option name and an *option_string*, specified in the section option_string on page 62.

The options you can specify for msgsend are in Table 4-5 on page 41.

properties_clause

is either an *option_string* or one of the options listed in the following tables. The options described in Table 4-5 on page 41 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 4-6 on page 41, it is set as a property in the message properties of the message sent.

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
is a select expression that specifies a for xml clause.

header_clause
allows users to specify only header properties You see an error if you enter an unrecognized header property.

If a recognized header property is specified in both the *message property* and the *message header* clauses, the one in the *message header* clause takes precedence.

You get an error when you specify any unrecognized options in the *option_clause*.

All previously recognized header properties are accepted in the *message header* clause.

Examples

To publish messages, you must define a subscription on the server to which the client is connected:

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

The client server can then publish a message to a specified subscription:

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

Usage

- Unrecognized options are ignored if you use message property. If you use message header for the msgsend or msgpublish functions, you see an error when you specify unrecognized options.
- 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'
  ```

- Table 4-5 lists the options you can specify for msgsend.

Table 4-5: option_string values for msgpublish

Option	Values	Default	Comments
schema	• no • yes • "*user_schema*"	no	Enter one of these values: • *user_schema* – is a user-supplied schema describing the message_body. • no – indicates that no schema is generated and sent out as part of the message. • yes – indicates that Adaptive Server generates an XML schema for the message. yes is meaningful only 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 or bytes	text	The message type to send.

- Table 4-6 lists the options and values for the *properties_clause* parameter. If you use a property not listed in Table 4-6, it is set as a property in the message properties of the message sent.

Table 4-6: Values for the msgpublish properties_clause parameter

Option	Values	Default	Disposition	Comments
correlation	*string*	none	header	Client applications set correlation IDs to link messages together. Adaptive Server sets the correlation ID the application specifies.
mode	• persistent • non-persistent	persistent	header	When you enter: • 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. • non-persistent and the messaging provider crashes – you may lose a message before it reaches the desired destination.
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.

Option	Values	Default	Disposition	Comments
replyqueue	A string containing a *queue_name*	none	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.
replytopic	A string containing a *topic_name*	none	header	
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 indicates that the life of the message is 60 milliseconds. A value of 0 indicates that the message never expires.

Permissions

You must have messaging_role to run msgpublish.

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
 message_filter ::= basic_character_expression
```

Parameters

basic_character_expression
is a SQL query expression with a datatype is char, varchar, or java.lang.String.

end_point
is a *basic_character_expression* where the runtime value is a *service_provider_uri*. The destination of a message.

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

message_filter
is 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.

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.

option
is a value shown in Table 4-7 on page 45.

option_string
is the general format of the *option_string* is specified in option_string on page 62. The options for msgrecv are described in Table 4-7 on page 45.

returns_clause
is 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

Examples

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

```
select msgrecv
     ('tibco_jms:tcp://my_ems_host:7222?queue=queue.sample,'
     +'user=ems_user1,password=ems_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_ems_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_ems_provider?queue=queue.sample'),
     'another_ems_provider?queue=queue2')
```

Example 4 Calls only consumes messages from queue.sample when the message property “Name” is equal to “John Smith”:

```
select msgrecv('my_ems_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_ems_provider?queue=queue.sample',
     RETURNS text)
```

Example 6 Reads a message and returns it as a varbinary.

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

Usage

- Table 4-7 on page 45 lists the available *option* and *option_string* values for properties of msgrecv.

Table 4-7: option and option_string values for msgrecv

option **values**	***option_string*** **values**	**Default**	**Description**
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.
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 a null value when the timeout interval lapses without reading a message. The values are in numbers of milliseconds.

- Unrecognized option names result in an error.
- See section "@@msgheader" on page 14 regarding properties read from the message header.
- 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 a null value 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 you use requeue, messages that Adaptive Server cannot handle are placed on the specified queue.

  The specified endpoint must exist on the same message 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.
- 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>* and *<msgproperties>* document with msgpropvalue. For details, see msgpropvalue on page 37.
  - The general format of *<msgheader>* and *<msgproperties>* is described in "Messaging-related global variables" on page 13.

Quoting property or option values

- Place 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 true only 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_ems_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_ems_provider?queue=queue.sample',
     MESSAGE SELECTOR "color='red'")
```

In this next 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_ems_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_ems_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_ems_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_ems_provider?queue=queue.sample',
     MESSAGE PROPERTY 'color=red, shape=square'
```

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

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

Message filters

- 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.

- If *message_filter* is specified to msgrecv, it is ignored.

Permissions

You must have messaging_role to run msgrecv.

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]
        [header_clause]
option_clause ::= [,] option option_string
properties_clause ::= [,] message property property_option_string
header_clause ::= [,] message header header_option_string
message_body ::= scalar_expression | (select_for_xml)
end_point ::= basic_character_expression
```

Parameters

message_body

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

endpoint

is the queue to which a message is addressed. *endpoint* is a *basic_character_expression* where the runtime value is a *service_provider_uri*.

option

allows you to specify options for msgsend. Use the options in Table 4-8 if you are using TIBCO.

option_string

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

```
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
```

Parameter	Description
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.

Parameter	**Description**
false	A Boolean literal.
null	A null literal.
byte_literal	Has the form 0xHH, where each H is a hexadecimal digit.

properties_clause

is a *property_option_string*, or one of the options listed in Table 4-9 on page 53 for TIBCO EMS. The options described in these two tables 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.

Use the options in Table 4-9 on page 53 for msgsend. If you use a property not listed in Table 4-9, it is set as a property in the message properties of the message sent.

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 with datatype that is char, varchar, or java.lang.String.

select_for_xml

A select expression that specifies a for xml clause.

header_clause

allows users to specify only those header properties that are specified in d Table 4-9 on page 53. You see an error if you enter an unrecognized header property.

If a recognized header property is specified both in the *message property* and the *message header* clauses, the one in the *message header* clause takes precedence.

You get an error when you specify any unrecognized names in the *message header* parameter.

Examples

Example 1 Sends the message “Hello” to the specified endpoint:

```
select msgsend('Hello', 'my_ems_provider?queue=queue.sample,'
    +'user=ems_user1,password=ems_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_ems_provider?queue=queue.sample,user=ems_user1,'
    +'password=ems_user1_password')
```

Example 3 Sends a message with a body that 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_ems_host:7222?queue=queue.sample,'
     +'user=ems_user1,password=ems_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_ems_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_ems_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

- 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 4-2 on page 17.
- 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'`; other values are ignored:

```
select msgsend( 'Hello Messaging World!',
     'my_ems_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 “Messaging-related global variables” on page 13.
- 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.

- Unrecognized options or properties are ignored, but unrecognized option or property values are flagged as an error.

msgsend option *option_string* parameter values

Table 4-8 lists the available msgsend option parameters for TIBCO.

Table 4-8: Valid TIBCO EMS option option_string types and values for msgsend

Types	Values	Default	Description
schema	• no • yes • "*user_schema*"	no	• *user_schema* is a user-supplied schema describing the *message_body*. • no indicates that no schema is generated and sent out as part of the message. • yes indicates that Adaptive Server generates an XML schema for the message. yes is meaningful only 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.

msgsend *properties_clause* parameter values

Table 4-9 lists the available msgsend *properties_clause* parameters for TIBCO EMS.

Table 4-9: Valid message property properties_option_string types and values for msgsend

Option	Values	Default	Disposi-tion	Description
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 is the duration of time during which a message is valid, in milliseconds. For instance, 60 indicates that the life of the message is 60 milliseconds. A value of 0 indicates that the message never expires.
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.

Option	Values	Default	Disposi-tion	Description
mode	• persistent • non-persistent	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 is consumed, the message is lost, unless mode is set to persistent. • non-persistent and the messaging provider crashes – you may lose a message before it reaches the desired destination.
replyqueue	A string containing a *queue_name*	none	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.
replytopic	A string containing a *topic_name*	none	header	

msgsend properties and *rfhCommand*

- Unrecognized options are ignored if you use message property.
- 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 only 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.

Permissions

You must have messaging_role to run msgsend.

msgsubscribe

Description

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

Syntax

msg_subscribe::= msgsubscribe
    (*subscription_name*)

subscription_name::=*basic_character_expression*

Parameters

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

Examples

Tells the JMS messaging provider to begin holding messages published to the topic registered as "subscription_1":

```
select msgsubscribe ('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_ems_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 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.

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 4-10: 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 TIBCO 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. It is then 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')
```

msgunsubscribe

Description

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

Syntax

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

subscription_name::=*basic_character_expression*

Parameters

subscription_name
is 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 EMS message provider.

Examples

Tells the TIBCO 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_ems_provider?topic=topic.sample,user=user1,password=pwd',
     'Supplier=12345', null, 'durable1', 'client1'
```

- 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 EMS 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 4-11: 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 EMS 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')
  ```

option_string

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
is the string describing the option you want to specify.

simple_identifier
is the string that identifies the value of an *option*.

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

integer_literal
is the literal specified by normal SQL conventions.

float_literal
is the literal specified by normal SQL conventions.

true
is a Boolean literal.

false
is a Boolean literal.

null
is 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 49.

CHAPTER 5 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, a failed messaging operation using 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 invisible 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 set transactional messaging command, which 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 is sent 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 6 Samples

This chapter describes sample code illustrating messaging functionality that is distributed with Adaptive Server Real-Time Data Services (RTDS).

Sybase directories

The SYBASE directory contains three subdirectories:

- *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 RTDS.
- *jdbc* – JDBC samples using RTDS.

You can find the code samples in the *$SYBASE/$SYBASE_ASE/samples/messaging* directory.

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 are not named exactly the same. For example, *queue_listener.bat* on a Windows platform may be *queue_listener* on a UNIX or Linux platform.

Using code samples with Replication Server function strings

These code samples assume that you have some basic knowledge of Replication Server setup and configuration, as well as 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 Adaptive Server version 12.5.1 and earlier that does not support for messages directly 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 how to use 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.

Glossary

The terms defined here are used throughout this document.

Durable subscription	A subscription that retains messages while the client is not connected.
EMS	Enterprise Message Service, the TIBCO implementation of a Java messaging service.
JMS	A 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	A message provider such as TIBCO EMS.
Subscription	A domain for publishing or consuming one-to-many messaging.
Topic	Similar to queues in, but used for one-to-many messaging.

Index