DOCUMENT ID: DC20154-01-1253-01

LAST REVISED: July 2005

Copyright © 1987-2005 by Sybase, Inc. All rights reserved.

This publication pertains to Sybase software and to any subsequent release until otherwise indicated in new editions or technical notes. Information in this document is subject to change without notice. The software described herein is furnished under a license agreement, and it may be used or copied only in accordance with the terms of that agreement.

To order additional documents, U.S. and Canadian customers should call Customer Fulfillment at (800) 685-8225, fax (617) 229-9845. Customers in other countries with a U.S. license agreement may contact Customer Fulfillment via the above fax number. All other international customers should contact their Sybase subsidiary or local distributor. Upgrades are provided only at regularly scheduled software release dates. No part of this publication may be reproduced, transmitted, or translated in any form or by any means, electronic, mechanical, manual, optical, or otherwise, without the prior written permission of Sybase, Inc.


02/05

Unicode and the Unicode Logo are registered trademarks of Unicode, Inc.

All other company and product names used herein may be trademarks or registered trademarks of their respective companies.

Use, duplication, or disclosure by the government is subject to the restrictions set forth in subparagraph (c)(1)(ii) of DFARS 52.227-7013 for the DOD and as set forth in FAR 52.227-19(a)-(d) for civilian agencies.

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

About This Book ............................................................................................................................ v

### CHAPTER 1

**Introduction** .............................................................................................................................. 1  
- RTMS messaging concepts ................................................................. 1  
- Automatic decisions in real time.......................................................... 2  
- Messaging models .......................................................................... 2  
  - JMS .......................................................................................... 3  
  - MQSeries messaging models ....................................................... 3  
- Message format ............................................................................. 4  
  - JMS message properties............................................................ 4  
  - MQ message topics ................................................................... 5  
- Message selectors ......................................................................... 5

### CHAPTER 2

**Understanding Real Time Messaging Services** ................................................................. 7  
- Sending and receiving messages from a queue ...................................... 7  
- Publishing and consuming messages from a JMS topic ..................... 8  
- Working with message properties ..................................................... 8  
- Previewing the messaging interface................................................... 9  
- MQSeries overview ........................................................................ 11  
- MQSeries publish/subscribe ............................................................ 13  
  - Syntax for topics ...................................................................... 16  
  - Subscriber and publisher identities ............................................. 18  
  - MQ publish/subscribe examples .................................................. 18  
- MQSeries security .......................................................................... 25  
  - Connecting to the MQ Queue Manager ...................................... 25  
  - Installing MQ client on Adaptive Server host machines .......... 25  
  - Installing MQ client libraries ..................................................... 25  
- MQ authorizations ......................................................................... 26

### CHAPTER 3

**Configuring Real Time Messaging Services** ................................................................. 27  
- Configuring RTMS ........................................................................ 27  
- Configuring Adaptive Server for MQ ............................................. 28  
  - Using sp_configure to configure the Q engine .......................... 28
## Contents

### CHAPTER 4 SQL Reference

- Message-related global variables ........................................... 32
- `<msgheader>` and `<msgproperties>` documents ....................... 38
- Adaptive Server-specific message properties .......................... 40
- Keywords ............................................................................... 41
- Stored procedures .................................................................... 42
- Built-in functions ..................................................................... 42
- Syntax segments ...................................................................... 43
- `sp_engine` ............................................................................. 44
- `sp_msgadmin` ....................................................................... 48
- `msgconsume` .......................................................................... 57
- `msgpropcount` ....................................................................... 60
- `msgproplist` .......................................................................... 61
- `msgpropname` ........................................................................ 63
- `msgproptype` ......................................................................... 64
- `msgpropvalue` ........................................................................ 67
- `msgpublish` .......................................................................... 69
- `msgrecv` ................................................................................. 73
- `msgsend` ............................................................................... 88
- `msgunsubscribe` ..................................................................... 124
- `endpoint` ............................................................................... 130
- `option_string` ....................................................................... 133
- `sizespec` ............................................................................... 134
- `timespec` ............................................................................... 135

### CHAPTER 5 Transactional Behavior

- Transactional message behavior .............................................. 137
- Transactional messaging set option .......................................... 137

### CHAPTER 6 Samples

- Sybase directories ..................................................................... 139
- Using code samples with Replication Server function strings ...... 140
- Using code samples with SQL .................................................. 140
- Using code samples with Java/JDBC ........................................ 140

### Glossary

- Index

---

Adaptive Server Enterprise
About This Book

Audience

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

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 Messaging Services,” is an overview of Real Time Messaging Services (RTMS) specific to Adaptive Server.
- Chapter 3, “Configuring Real Time Messaging Services,” provides a procedure for configuring your system.
- Chapter 4, “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


Related documents

The following documents make up the Sybase® Adaptive Server Enterprise documentation set:

Messaging Services User’s Guide
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 this installation and release bulletin may be available on the Web. To check for critical product or document information added after the release of the product CD, use the Sybase Technical Library Product Manuals Web site. To access the most recent release bulletin:

b. Follow the links to the appropriate Sybase product.
c. Select the Release Bulletins link.
d. Select the Sybase product version from the Release Bulletins list.
e. From the list of individual documents, select the link to the release bulletin for your platform. You can either download the PDF version or browse the document online.

The Installation Guide for your platform – describes installation, upgrade, and configuration procedures for all Adaptive Server and related Sybase products.

What's New in Adaptive Server Enterprise? – describes the new features in Adaptive Server version 12.5.3, the system changes added to support those features, and the changes that may affect your existing applications.

ASE Replicator User's Guide – describes how to use the ASE Replicator feature of Adaptive Server to implement basic replication from a primary server to one or more remote Adaptive Servers.

Component Integration Services User's Guide – explains how to use the Adaptive Server Component Integration Services feature to connect remote Sybase and non-Sybase databases.


Error Messages and Troubleshooting Guide – explains how to resolve frequently occurring error messages and describes solutions to system problems frequently encountered by users.
• **Full-Text Search Specialty Data Store User’s Guide** – describes how to use the Full-Text Search feature with Verity to search Adaptive Server Enterprise data.

• **Glossary** – defines technical terms used in the Adaptive Server documentation.


• **jConnect for JDBC Programmer’s Reference** – describes the jConnect™ for JDBC™ product and explains how to use it to access data stored in relational database management systems.

• **Job Scheduler User’s Guide** – provides instructions on how to install and configure, and create and schedule jobs on a local or remote Adaptive Server using the command line or a graphical user interface (GUI).

• **Monitor Client Library Programmer’s Guide** – describes how to write Monitor Client Library applications that access Adaptive Server performance data.


• **Performance and Tuning Guide** – is a series of four books that explains how to tune Adaptive Server for maximum performance:
  
  • **Basics** – the basics for understanding and investigating performance questions in Adaptive Server.

  • **Locking** – describes how the various locking schemas can be used for improving performance in Adaptive Server.

  • **Optimizer and Abstract Plans** – describes how the optimizer processes queries and how abstract plans can be used to change some of the optimizer plans.

  • **Monitoring and Analyzing** – explains how statistics are obtained and used for monitoring and optimizing performance.

• **Quick Reference Guide** – provides a comprehensive listing of the names and syntax for commands, functions, system procedures, extended system procedures, datatypes, and utilities in a pocket-sized book.

• **Reference Manual** – is a series of four books that contains the following detailed Transact-SQL® information:
• **Building Blocks** – Transact-SQL datatypes, functions, global variables, expressions, identifiers and wildcards, and reserved words.

• **Commands** – Transact-SQL commands.

• **Procedures** – Transact-SQL system procedures, catalog stored procedures, system extended stored procedures, and dbcc stored procedures.

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

• **System Administration Guide** – provides in-depth information about administering servers and databases. This manual includes instructions and guidelines for managing physical resources, security, user and system databases, and specifying character conversion, international language, and sort order settings.

• **System Tables Diagram** – illustrates system tables and their entity relationships in a poster format. Available only in print version.

• **Transact-SQL User's Guide** – documents Transact-SQL, Sybase's enhanced version of the relational database language. This manual serves as a textbook for beginning users of the database management system. This manual also contains descriptions of the pubs2 and pubs3 sample databases.

• **Using Adaptive Server Distributed Transaction Management Features** – explains how to configure, use, and troubleshoot Adaptive Server DTM features in distributed transaction processing environments.

• **Using Sybase Failover in a High Availability System** – provides instructions for using Sybase Failover to configure an Adaptive Server as a companion server in a high availability system.

• **Utility Guide** – documents the Adaptive Server utility programs, such as isql and bcp, which are executed at the operating system level.

• **Web Services User's Guide** – explains how to configure, use, and troubleshoot Web Services for Adaptive Server.

• **XA Interface Integration Guide for CICS, Encina, and TUXEDO** – provides instructions for using the Sybase DTM XA interface with X/Open XA transaction managers.

• **XML Services in Adaptive Server Enterprise** – describes the Sybase native XML processor and the Sybase Java-based XML support, introduces XML in the database, and documents the query and mapping functions that comprise XML Services.
Other sources of information

Use the Sybase Getting Started CD, the 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

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.

2. Click MySybase and create a MySybase profile.

Finding the latest information on EBFs and software maintenance

2. Select EBFs/Maintenance. If prompted, enter your MySybase user name and password.
3. Select a product.
4. Specify a time frame and click Go. A list of EBF/Maintenance releases is displayed.
   Padlock icons indicate that you do not have download authorization for certain EBF/Maintenance releases because you are not registered as a Technical Support Contact. If you have not registered, but have valid information provided by your Sybase representative or through your support contract, click Edit Roles to add the “Technical Support Contact” role to your MySybase profile.
5. Click the Info icon to display the EBF/Maintenance report, or click the product description to download the software.

Conventions

In the regular text of this document, the names of files and directories appear in italics, for example:
• In Windows NT: `%SYBASE%\bin
• In UNIX platforms: `$SYBASE`

**Note** Substitute your Sybase installation drive and directory for `$SYBASE` in UNIX, and `%SYBASE%` in Windows NT.

Table 1 details the typographic (font and syntax) conventions as used in this document.

<table>
<thead>
<tr>
<th><strong>Table 1: Font and syntax conventions for this document</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Element</strong></td>
</tr>
<tr>
<td>Command names, command option names, database names, datatypes, utility names, utility flags, and other keywords are <strong>Helvetica</strong>.</td>
</tr>
<tr>
<td>Variables, or words that stand for values that you fill in, are in <strong>italics</strong>.</td>
</tr>
<tr>
<td>Parentheses must be typed as part of the command.</td>
</tr>
<tr>
<td>Curly braces indicate that at least one of the enclosed options is required by the command (see comma).</td>
</tr>
<tr>
<td><strong>Note</strong> Do not type the curly braces.</td>
</tr>
<tr>
<td>Brackets mean that choosing one or more of the enclosed options is optional.</td>
</tr>
<tr>
<td><strong>Note</strong> Do not type the brackets.</td>
</tr>
<tr>
<td>The vertical bar means you may select only one of the options shown.</td>
</tr>
<tr>
<td><strong>Note</strong> Do not type the curly braces.</td>
</tr>
<tr>
<td>The comma means you may choose as many of the options shown as you like; be sure to separate multiple choices in a command with commas.</td>
</tr>
<tr>
<td><strong>Note</strong> Do not type the brackets.</td>
</tr>
<tr>
<td>An ellipsis (...) means that you can <strong>repeat</strong> the unit that the ellipsis follows as many times as you like.</td>
</tr>
<tr>
<td>• You must buy at least one <code>thing</code> (item) and give its price.</td>
</tr>
<tr>
<td>• You may choose a method of payment: one of the options enclosed in square brackets.</td>
</tr>
<tr>
<td>• You may choose also to buy additional items: as many of them as you like. For each item you buy, provide its name, its price, and (optionally) a method of payment.</td>
</tr>
</tbody>
</table>
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.
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 Messaging Services (RTMS).

### RTMS messaging concepts

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

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

Real Time Messaging Services (RTMS), simplifies the development of such applications, using Adaptive Server with TIBCO Java Messaging Service (JMS) and IBM WebSphere MQSeries (MQ).

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 and MQSeries 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.

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

Real Time Messaging Services provide a way to capture transactions (data changes) in an Adaptive Server database and deliver them as events to external applications using either:

- JMS message bus, provided by TIBCO Enterprise for JMS, or
- Message Queue Interface (MQI), provided by WebSphere MQSeries.

**Automatic decisions in real time**

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

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

These business needs are addressed by Sybase Real Time Data Services (RTDS) and RTMS, using the TIBCO JMS message bus or IBM WebSphere MQ.

**Messaging models**

This section describes the messaging models for JMS and MQ Series.
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.

MQSeries messaging models

All MQ messaging models are point-to-point, that is, messages are always sent to, or received from a queue that is managed by a queue manager.
MQ pub/sub is a publish-and-subscribe model built on MQ queues; the messages are not different types of objects. Interaction with MQ pub/sub uses MQ queues.

All messages are sent to the MQ pub/sub broker’s broker command queue. This includes registration of a publisher or subscriber, and control messages such as deleting a message, or requesting an update for a message.

A publisher sends a publication to a stream queue. The MQ pub/sub broker distributes the message to all subscribers that have interest in the message. The publisher describes the message using topics, which are subjects that describe the contents of the message.

Subscribers register interest in messages that are sent to a named stream queue by specifying one or more topics of interest. When such messages are sent to the stream queue, the MQ pub/sub broker copies the message to the local queue that the subscriber specified when the subscriber was registered.

Message format

The message format for both MQ and 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.

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

In MQSeries, the message body can contain both text and binary data.

JMS message properties

In TIBCO, 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).
MQ message topics

The MQ, the pub/sub model allows “topics,” which are the subjects of messages. The topics are included in the message in the rules and formatting (RF) header. Unlike JMS, MQ topics are not name-value pairs—which consist of a name and its accompanying value—but are free-form strings that describe the MQ pub/sub message.

Message selectors

TIBCO JMS – message selectors for TIBCO JMS 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.

MQSeries – message selection uses only the message ID and message correlation ID as message selectors. A message reader can selectively choose to read a particular message by specifying a message ID or message correlation ID.
Message selectors
CHAPTER 2
Understanding Real Time Messaging Services

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

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sending and receiving messages from a queue</td>
<td>7</td>
</tr>
<tr>
<td>Publishing and consuming messages from a JMS topic</td>
<td>8</td>
</tr>
<tr>
<td>Working with message properties</td>
<td>8</td>
</tr>
<tr>
<td>Previewing the messaging interface</td>
<td>9</td>
</tr>
<tr>
<td>MQSeries overview</td>
<td>11</td>
</tr>
<tr>
<td>MQSeries publish/subscribe</td>
<td>13</td>
</tr>
<tr>
<td>MQSeries security</td>
<td>25</td>
</tr>
<tr>
<td>Installing MQ client on Adaptive Server host machines</td>
<td>25</td>
</tr>
<tr>
<td>MQ authorizations</td>
<td>26</td>
</tr>
</tbody>
</table>

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 in JMS and MQSeries.

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.
Publishing and consuming messages from a JMS topic

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

Message properties in read messages can be individually processed by the application logic. For more information about message properties, see msgsend on page 88.

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 n\textsuperscript{th} 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
- msgpropctype
- msgpropcount

**Previewing the messaging interface**

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

**Examples**

**Example 1**  JMS – sends a message to a queue:

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

**Example 2**  JMS – reads a message from a queue, with and without a filter:

```sql
select msgrecv('tibco_jms:tcp://my_jms_host:7222?queue=queue.sample')
select msgrecv ('tibco_jms:tcp://my_jms_host:7222?queue=queue.sample'
               message selector 'city=''Detroit''')
```

**Example 3**  JMS – publishes a message to a topic:

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

**Example 4**  JMS – consumes a message from a topic:
select msgconsume('sub1')

Example 5  JMS – 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 @curr=@curr+1
    select @name=msgpropvalue(@pname)
end

Example 6  MQSeries – sends a message to a queue:

    select msgsend('hello world',
    'ibm_mq:chnl1/tcp/host1(1234)?qmgr=QM,queue=DEFAULT.QUEUE'
    message header 'priority=2')

Example 7  MQSeries – reads a message from a queue:

    select msgrecv(
    'ibm_mq:chnl1/tcp/host1(1234)?qmgr=QM,queue=DEFAULT.QUEUE'
    option 'timeout=30ss')

Example 8  MQSeries – registers a publisher and publishes a message about “fish”:

    select msgsend(NULL,
    'ibm_mq:chnl1/tcp/host1(1234)?qmgr=QM,queue=SYSTEM.BROKER.CONTROL.QUEUE'
    option 'rfhCommand=registerPublisher'
    message header 'topics=fish'
    + ',streamName=ANIMALS.STREAM')
select msgsend('something about a fish',
    'ibm_mq:chnl1/tcp/host1(1234)?qmgr=QM,queue=ANIMALS.STREAM'
    message header 'topics=fish')

Example 9  MQSeries – registers a subscriber, reads a message, and processes the message properties:

    select msgsend(NULL,
    'ibm_mq:chnl1/tcp/host1(1234)?qmgr=QM,queue=SYSTEM.BROKER.CONTROL.QUEUE'
    option 'rfhCommand=registerSubscriber'
    + ',topics=fish'
    + ',streamName=ANIMALS.STREAM'
    + ',queueName=MY_ANIMALS.QUEUE')
MQSeries overview

IBM WebSphere MQSeries allows different applications to communicate asynchronously through queues across different operating systems, different processors, and different application systems.

WebSphere MQSeries includes the Message Queue Interface (MQI), a common low-level programming application program interface (API). Applications use MQI to read and write messages to the queues.

A Queue Manager is a process that manages a set of objects. These objects include queues, channels, and process definitions.

A queue object stores messages that are sent by applications to the Queue Manager. The following are types of queues:

- A local queue – is owned by the Queue Manager to which a program is connected.
- A remote queue – is owned by a Queue Manager other than the Queue Manager to which a program is connected. A remote queue can be sent messages, but messages cannot be read from it.
- An alias queue – is another name for a local or remote queue.
- A dynamic local queue – is a queue that is created on the fly by an application. It is created from a model queue. The persistence of a dynamic queue is defined by the model queue from which it is created.
- A model queue – is a queue that is used as a template for creating a dynamic local queue.
- A cluster queue – is a queue that is owned by a cluster queue manager.
- A channel – is a logical communication link. Channel types are client (client side of a connection), or server (server side of a connection).

A process definition defines a process that executes when incoming messages cause a trigger event.
A WebSphere MQSeries message consist of two parts:

- Message header – message control information that contains a fixed-sized portion and a variable-sized portion.
- Message body – application data that contains any type of data (text or binary).

When you use rfhCommand to publish a publication, if the message payload returned by msgrecv is set to:

- MQRHRF – the RF header is included in the message body.
- MQRHRH – the RF header is not included.

You can obtain the name value pairs in the RF header by querying @@msgproperties.

If the message body contains characters, code-set conversions are available either through MQSeries native services, or through user exit handlers. The format of the message body is defined by a field in the message header. MQ does not enumerate all possible message body formats, although some formats are provided in samples. Applications can enter any name of the format. For instance, “MQSTR” contains string data, “MQRHRF” contains topics for MQ pub/sub.

WebSphere MQSeries message types include the following

- Datagram – no reply is expected.
- Request – a reply is expected.
- Reply – reply to a request message.
- Report – contains status information from the Queue Manager or another application.

When messages are sent, various message header properties can be set, such as expiration, persistence, priority, correlation ID, and reply queue.

Message grouping enables you to organize a group of messages into a logically named group. Within a group, each logical message can further be divided into segments. A group is identified by a name, each logical message within a group is identified by a sequence number (starting with 1), and each segment of a logical message is identified by the offset of the message data with respect to the logical message. Segmented messages are not supported by MQ pub/sub, and an attempt to send a segmented message results in an error.
In a queue, messages appear in the physical order in which they were sent to the queue. This means that messages of different groups may be interspersed, and, within a group, the sequence numbers of the messages may be out of order (the latter can occur if two applications are sending messages with the same group ID and partitioned sequence numbers).

When messages are received, the read mode can be either:

- Destructive – message is removed, or
- Nondestructive – the message is retained. This is known as “browsing,” and allows applications to peruse one or more messages before deciding to remove a particular message from the queue.

Receivers can select particular messages by specifying message header properties such as correlation ID or message ID.

When messages are read—as either destructive or nondestructive—the order in which they are returned can be physical or logical. The order is defined by the queue definition. The queue can be defined as being in priority order or first-in, first-out order.

**MQSeries publish/subscribe**

WebSphere MQSeries publish/subscribe is on MQSeries queues that employ a broker process to perform subscription resolution. In its simplest form:

- A publisher is the application that is sending the message.
- A subscriber is the application that is receiving the message.
- The following three queues are involved:
  - Control queue – where publishers and subscribers send directives to the pub/sub broker. For instance subscriber registration and deregistration.
  - Stream queue – where the publisher sends its messages directly. The pub/sub broker reads the messages from the stream queue and distributes them to the appropriate subscriber’s queue.
Subscriber queue – where the subscriber reads its messages directly.

Note More queues can be involved, depending on the type of publications.

- The pub/sub broker responds to MQRFH messages sent to the control queue. These command messages control how the pub/sub broker processes messages that arrive on the stream queue. For instance, a subscriber could register an interest in a particular topic.
- The publisher sends messages directly to the stream queue.
- The pub/sub broker reads messages from the stream queue and determines the subscriber queue to which to copy the message. This depends on topics that the subscribers have registered interest in.
- The subscriber reads messages directly from the subscriber queue.

Subscribers register “subscriptions,” which means it is interested in one or more “topics”.

Example

This example, which shows the MQ pub/sub process, uses these variables:

declare @BROKER varchar(100)
declare @STREAM varchar(100)
declare @SUBQ varchar(100)
declare @QM varchar(100)
select @QM = 'ibm_mq:chn1/tcp/host1(9876)?qmgr=QM'
select @BROKER = 'SYSTEM.BROKER.CONTROL.QUEUE'
select @STREAM = 'ANIMALS'
select @SUBQ = 'MY_ANIMALS'

1 Publisher registers to send publications to ANIMALS with topics on fish:
select msgsend(NULL,
    @QM + ',queue=' + @BROKER
    option 'rfhCommand=registerPublisher'
    message header 'topics=fish,streamName=' + @STREAM)

2 Subscriber registers to receive publications published to ANIMALS with topics on fish. The subscriber receives the publications on MY_ANIMALS:
select msgsend(NULL,
    @QM + ',queue=' + @BROKER
    option 'rfhCommand=registerSubscriber'
    message header 'topics=fish'
    + ',streamName=' + @STREAM)
Publisher publishes publication to ANIMALS about fish. The MQ pub/sub broker automatically forwards the publication to MY_ANIMALS:

```sql
select msgsend('something about fish',
               @QM + ',queue=' + @STREAM
               option 'rfhCommand=publish'
               message header 'topics=fish')
```

Subscriber reads the forwarded message from MY_ANIMALS:

```sql
select msgrecv(@QM + ',queue=' + @SUBQ option 'timeout=30ss')
```

Figure 2-1 shows the flow of the sample MQ pub/sub process.

![Figure 2-1: The MQ publication/subscription process](image)

A message can have one or more topics. WebSphere MQSeries pub/sub recommends that topics use a hierarchical naming convention as in the examples show below. Subscribers can specify wildcards (such as * and ?) when specifying topics of interest.

These are examples of topics:

- Sport
- Sport/Soccer
- Sport/Tennis

These are examples of how subscribers can specify topics of interest:

- Sport/* - Any topic about sports.
MQSeries publish/subscribe

/**Soccer - Any topics about soccer.
/**Soccer/Trades - Any topics about soccer where a 'trade' is involved.

A retained publication is a type of publication where the MQ pub/sub broker maintains a copy of a message even after it has delivered it to all subscribers. Normally, a publication is deleted after a copy has been delivered to all subscribers. A retained publication allows a subscriber to asynchronously request the retained publication instead of relying on it being delivered by the MQ pub/sub broker. These types of messages normally contain state information, and are also referred to as state publications.

Syntax for topics

- A topic is generally in the form “topic/subtopic,” for example “sport/baseball.”
- You can specify a wildcard, such as “*” or “?” within a topic.
- When specifying multiple topics, separate the topics with a colon. For instance, “topic1:topic2:topic3:..”.
- If a topic contains spaces or commas, the entire topic list must be placed in quotes. Since topics can appear in message header or message property clauses as strings, if the option string is passed as a quoted scalar value, the enclosed quotes must be escaped by doubling them. Furthermore, if the topic is contains also embedded double quotes, the embedded double quotes must be escaped by quadruple quotes. For example:

```sql
-- Topic has embedded spaces, we need to quote with escaped quotes
select msgsend(NULL,
    'ibm_mq:channel1/tcp/host1(5678)?qmgr=QM1,queue=SAMPLE.STREAM'
    option 'msgType=datagram,rfhCommand=publish'
message property 'topics=''Sport/Football/Hometown Bulldogs''')

-- Topic has embedded spaces, we can quote with double quotes
select msgsend(NULL,
    'ibm_mq:channel1/tcp/host1(5678)?qmgr=QM1,queue=SAMPLE.STREAM'
    option 'msgType=datagram,rfhCommand=publish'
message property 'topics="Sport/Football/Hometown Bulldogs"')

-- Topic has embedded spaces and embedded double quotes, the inner
-- double quotes need to be escaped.
set quoted_identifier off
select msgsend(NULL,
CHAPTER 2  Understanding Real Time Messaging Services

'sib_mq:channel1/tcp/host1(5678)?qmgr=QM1,queue=SAMPLE.STREAM'
  option 'msgType=datagram,rfhCommand=publish'
message property 'topics="quoted ""topic" here""

-- Topic has embedded spaces and embedded double quotes, double the
-- quotes around the topic, and quadruple the embedded quotes.
select msgsend(NULL,
  'ibm_mq:channel1/tcp/host1(5678)?qmgr=QM1,queue=SAMPLE.STREAM'
  option 'msgType=datagram,rfhCommand=publish'
message property "topics="quoted """"topic"" here""
)

- When topics have embedded spaces or quotes, the topic is quoted in the
  MQRF header. If the topic has embedded quotes, the quotes are escaped
  before being put into the MQRF header.

In the following example, there is one topic, which is placed in the MQRF
header as ""Sport/Football/Hometown Bulldogs"". Another example is:

select msgsend(NULL,
  'ibm_mq:channel1/tcp/host1(5678)?qmgr=QM1,queue=SAMPLE.STREAM'
  option 'msgType=datagram,rfhCommand=publish'
message property "topics="Sport/Football/Hometown Bulldogs"
)

In the following example, there is one topic, which is placed in the MQRF
header as "Books/""Recipes Of Spain""."".

select msgsend(NULL,
  'ibm_mq:channel1/tcp/host1(5678)?qmgr=QM1,queue=SAMPLE.STREAM'
  option 'msgType=datagram,rfhCommand=publish'
message property "topics="Books/"Recipes Of Spain"
)

You can escape topic name by using "::"; and any single, non-escaped
trailing "::" is ignored.

In the following example, there are three topics, "baseball",
"baseball/anytown", and "baseball/scores".

select msgsend(NULL,
  'ibm_mq:channel1/tcp/host1(5678)?qmgr=QM1,queue=SAMPLE.STREAM'
  option 'msgType=datagram,rfhCommand=publish'
message property 'topics=baseball:baseball/anytown:baseball/scores'
)

In this example, there are three topics, "subject1", "subject:2", and
"subject3". Note that "::" is used to escape the embedded "::".

select msgsend(NULL,
  'ibm_mq:channel1/tcp/host1(5678)?qmgr=QM1,queue=SAMPLE.STREAM'
  option 'msgType=datagram,rfhCommand=publish'
message property 'topics=subject1:subject::2:subject3'
)
Subscriber and publisher identities

By default, a publisher or subscriber identity consists of the following:

- Queue name.
- Queue manager name.
- Correlation identifier (optional). You can use the correlation identifier to distinguish between different publishers or subscribers using the same queue. Each publisher and subscriber can be assigned a different correlation identifier. This allows several applications to share a queue. It also allows a single application to differentiate publications originating from different subscriptions.

MQ publish/subscribe examples

Publisher example

The Adaptive Server session is a publisher. It publishes on “topicA” and “topicB”; publications on “topicB” are published as retained publications. The retained publication will be deleted.

```sql
-- @QM has the Queue Manager endpoint
declare @QM varchar(100)
-- @BROKER has the broker queue name
declare @BROKER varchar(100)
-- @STREAM has the stream queue name
declare @STREAM varchar(100)
-- @CORRELID has the generated correlation id
declare @CORRELID varchar(100)

-- Put Queue manager name, broker and stream queue names into variables
select @QM = 'ibm_mq:chan1/tcp/localhost(5678)?qmgr=QM1'
selcoot @BROKER = 'SYSTEM.BROKER.CONTROL.QUEUE'
selcoot @STREAM = 'Q1.STREAM'

-- Register the publisher, only for topicA
select msgsend(NULL, @QM + ',queue=' + @BROKER
   option 'rfhCommand=registerPublisher'
   message header 'correlationAsId=generate'
   + ',topics=topicA'
   + ',streamName=' + @STREAM)

0x414d51204652414e4349532e514d202041a3ebfb20014801

-- Save the generated correlation id
select @CORRELID = @@msgcorrelation
```

Adaptive Server Enterprise
-- Send two publications on topicA
select msgsend('topicA, publication 1', @QM + ',queue=' + @STREAM
  option 'rfhCommand=publish'
  message header 'correlationAsId=yes'
    + ',correlationId=' + @CORRELID
    + ',topics=topicA')
-----------------------------------------------------------------------
0x414d51204652414e4349532e514d202041a3ebfb20014803

select msgsend('topicA, publication 2', @QM + ',queue=' + @STREAM
  option 'rfhCommand=publish'
  message header 'correlationAsId=yes'
    + ',correlationId=' + @CORRELID
    + ',topics=topicA')
-----------------------------------------------------------------------
0x414d51204652414e4349532e514d202041a3ebfb20014805

-- Add another topic for this publisher
select msgsend(NULL, @QM + ',queue=' + @BROKER
  option 'rfhCommand=registerPublisher'
  message header 'correlationAsId=yes'
    + ',correlationId=' + @CORRELID
    + ',topics=topicB'
    + ',streamName=' + @STREAM)
-----------------------------------------------------------------------
0x414d51204652414e4349532e514d202041a3ebfb20014807

-- Publish a retained message on topicB
select msgsend('topicB, retained publication 1', @QM + ',queue=' + @STREAM
  option 'rfhCommand=pub'
  message header 'correlationAsId=yes'
    + ',correlationId=' + @CORRELID
    + ',topics=topicB'
    + ',retainPub=yes')
-----------------------------------------------------------------------
0x414d51204652414e4349532e514d202041a3ebfb20014809

-- Publish a second retained publication on topicB
-- This one will replace the current retained publication on topicB.
select msgsend('topicB, retained publication 2', @QM + ',queue=' + @STREAM
  option 'rfhCommand=pub'
  message header ',correlationAsId=Yes'
    + ',correlationId=' + @CORRELID
    + ',topics=topicB'
    + ',retainPub=yes')
**MQSeries publish/subscribe**

```sql
-- Delete the retained publication on topicB
select msgsend(NULL, @QM + ',queue=' + @STREAM
    option 'rfhCommand=deletePublication'
    message header 'topics=topicB'
    + ',streamName=' + @STREAM)
```

```sql
-- Deregister the publisher, for all topics.
select msgsend(NULL, @QM + ',queue=' + @BROKER
    option 'rfhCommand=deregisterPublisher'
    message header 'correlationAsId=yes'
    + ',correlationId=' + @CORRELID
    + ',deregAll=yes'
    + ',streamName=' + @STREAM)
```

---

**Subscriber example**

In this example, the Adaptive Server session subscribes to “topicA” and “topicB”; publications on “topicB” are published as retained publications. This subscriber processes retained publications by requesting an update from the pub/sub broker.

```sql
-- @QM has the Queue Manager endpoint
declare @QM varchar(100)
-- @BROKER has the broker queue name
declare @BROKER varchar(100)
-- @SUBQUEUE has the subscriber queue name
declare @SUBQUEUE varchar(100)
-- @STREAM has the stream queue name
declare @STREAM varchar(100)
-- @CORRELID has the generated correlation id
declare @CORRELID varchar(100)

-- Put broker and subscriber queue names into variables
select @QM = 'ibm_mq:chan1/tcp/localhost(5678)?qmgr=QM1'
select @BROKER = 'SYSTEM.BROKER.CONTROL.QUEUE'
select @SUBQUEUE = 'Q1.SUBSCRIBER'
select @STREAM = 'Q1.STREAM'

-- Register the subscriber, only for topicA
select msgsend(NULL, @QM + ',queue=' + @BROKER
    option 'rfhCommand=registerSubscriber'
```
message header 'correlationAsId=generate'
   + ',topics=topicA'
   + ',streamName=' + @STREAM
   + ',queueName=' + @SUBQUEUE)

-----------------------------------------------------------------------
0x414d51204652414e4349532e514d202041a3ebfb20014801
-- Save the generated correlation id
select @CORRELID = @@msgcorrelation

-- Add another topic for this subscriber
-- we will explicitly request update for publications on this topic.
select msgsend(NULL, @QM + ',queue=' + @BROKER
   option 'rfhCommand=registerSubscriber'
   message header 'CorrelationAsId=yes'
   + ',correlationId=' + @CORRELID
   + ',topics=topicB'
   + ',streamName=' + @STREAM
   + ',queueName=' + @SUBQUEUE
   + ',pubOnReqOnly=yes')

-----------------------------------------------------------------------
0x414d51204652414e4349532e514d202041a3ebfb20014803
-- The publisher now publishes messages in the following order:
-- topicA, topicB (*), topicA, topicB (*)
-- ( '*' denotes a retained publication )

-- Get the first message on the subscriber queue, it will be on topicA.
select msgrecv(@QM + ',queue=' + @SUBQUEUE option 'timeout=30ss')

-----------------------------------------------
publication on topicA

-- Get the second message on the subscriber queue, it will be on topicA.
select msgrecv(@QM + ',queue=' + @SUBQUEUE option 'timeout=30ss')

-----------------------------------------------
publication on topicA

-- Request the broker to now send retained publications on topicB
select msgsend(NULL, @QM + ',queue=' + @BROKER
   option 'rfhCommand=requestUpdate'
   message header 'CorrelationAsId=yes'
   + ',correlationId=' + @CORRELID
   + ',topics=topicB'
   + ',streamName=' + @STREAM
   + ',queueName=' + @SUBQUEUE)
MQSeries publish/subscribe

Broker response example

This example shows how can use request/response messaging to check the response from the pub/sub broker. A subscription is registered by user1, and the pub/sub broker response is checked. The same subscription is then registered again by user2, with a different subscription name, which causes an error response from the pub/sub broker.

Queries executed by user1:

-- @QM has the Queue Manager endpoint
declare @QM varchar(100)
-- @BROKER has the broker queue name
declare @BROKER varchar(100)
-- @SUBQUEUE has the subscriber queue name
declare @SUBQUEUE varchar(100)
-- @REPLY has the reply queue name
declare @REPLY varchar(100)

-- Put broker, subscriber and reply queue names into variables
select @QM     = 'ibm_mq:chan1/tcp/localhost(5678)?qmgr=QM1'
select @BROKER = 'SYSTEM.BROKER.CONTROL.QUEUE'
sel ect @SUBQUEUE = 'Q1.SUBSCRIBER'
select @REPLY    = 'Q1.REPLY'

-- Get the next message on the subscriber queue, it will be on topicB.
select msgrecv(@QM + ',queue=' + @SUBQUEUE option 'timeout=30ss')
-- publication on topicB

-- Get the next message on the subscriber queue, it will be on topicB.
select msgrecv(@QM + ',queue=' + @SUBQUEUE option 'timeout=30ss')
-- publication on topicB

-- Deregister the subscriber, for all topics.
select msgsend(NULL, @QM + ',queue=' + @BROKER
    option 'rfhCommand=deregisterSubscriber'
    message header 'CorrelationAsId=yes'
    + ',correlationId=' + @CORRELID
    + ',deregAll=yes'
    + ',streamName=' + @STREAM
    + ',queueName=' + @SUBQUEUE)
-- publication on topicB

Broker response example

This example shows how can use request/response messaging to check the response from the pub/sub broker. A subscription is registered by user1, and the pub/sub broker response is checked. The same subscription is then registered again by user2, with a different subscription name, which causes an error response from the pub/sub broker.

Queries executed by user1:

-- @QM has the Queue Manager endpoint
declare @QM varchar(100)
-- @BROKER has the broker queue name
declare @BROKER varchar(100)
-- @SUBQUEUE has the subscriber queue name
declare @SUBQUEUE varchar(100)
-- @REPLY has the reply queue name
declare @REPLY varchar(100)

-- Put broker, subscriber and reply queue names into variables
select @QM     = 'ibm_mq:chan1/tcp/localhost(5678)?qmgr=QM1'
select @BROKER = 'SYSTEM.BROKER.CONTROL.QUEUE'
sel ect @SUBQUEUE = 'Q1.SUBSCRIBER'
select @REPLY    = 'Q1.REPLY'
-- Register the subscriber.
select msgsend(NULL, @QM + ',queue=' + @BROKER
   option 'rfhCommand=registerSubscriber, msgType=request'
   message header 'correlationAsId=generate'
     + ',topics=topicA'
     + ',streamName=Q1.STREAM'
     + ',queueName=Q1.SUBSCRIBER'
     + ',replyToQueue=Q1.REPLY')
-----------------------------------------------------------------------
0x414d51204652414349532e514d202041a3ebfb20014801
-- Read the response
select msgrecv(@QM + ',queue=' + @REPLY option 'timeout=30ss')
-----------------------------------------------------------------------
NULL
-- Check @@msgproperties
select @@msgproperties
-----------------------------------------------------------------------
<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
<msgproperties
   MQPSReasonText="&apos;MQRC_NONE&apos;"
   MQPSReason="0"
   MQPSCompCode="0">
</msgproperties>
-- Check MQPSCompCode
if (msgpropvalue('MQPSCompCode', @@msgproperties) != "0")
begin
print "registerSubscriber failed"
end

Queries executed by user2:
-- @QM has the Queue Manager endpoint
declare @QM varchar(100)
-- @BROKER has the broker queue name
declare @BROKER varchar(100)
-- @SUBQUEUE has the subscriber queue name
declare @SUBQUEUE varchar(100)
-- @REPLY has the reply queue name
declare @REPLY varchar(100)

-- Put broker, subscriber and reply queue names into variables
select @QM= 'ibm_mq:chan1/tcp/localhost(5678)?qmgr=QM1'
select @BROKER= 'SYSTEM.BROKER.CONTROL.QUEUE'
MQSeries publish/subscribe

```
select @SUBQUEUE= 'Q1.SUBSCRIBER'
select @REPLY= 'Q1.REPLY'
-- Register the subscriber
select msgsend(NULL, @QM + ',queue=' + @BROKER
    option 'rfhCommand=registerSubscriber, msgType=request'
    message header 'correlationAsId=generate'
    + ',topics=topicA'
    + ',streamName=Q1.STREAM'
    + ',queueName=Q1.SUBSCRIBER'
    + ',replyToQueue=Q1.REPLY')

-- Read the response
select msgrecv(@QM + ',queue=' + @REPLY option 'timeout=30ss')

-- Check @@msgproperties
select @@msgproperties

<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
<msgproperties
    MQPSUserId="&apos;user2 &apos;"
    MQPSReasonText="&apos;MQRCCF_DUPLICATE_IDENTITY&apos;"
    MQPSReason="3078"
    MQPSCompCode="2"
</msgproperties>

-- Check MQPSCompCode
if (msgpropvalue('MQPSCompCode', @@msgproperties) != "0")
begin
    print "registerSubscriber failed"
end
```
MQSeries security

Connecting to the MQ Queue Manager

You cannot specify a username and password with the MQ endpoint as you can using TIBCO JMS. All connections to the MQ Queue Manager are made as the user that the Adaptive Server process is running as. After making the connection to the MQ Queue Manager, Adaptive Server then attempts to open the queue as the Adaptive Server login that is performing the operation. For this reason, the user must:

- Have a user account on the machine on which MQ Queue Manager is running
- Be an MQ user
- Have the MQ authorizations listed in Table 2-3 on page 26.

Note The Adaptive Server “messaging_role” is still required to execute Real Time Data Services built-in functions.

In addition, the ‘register, login’ and ‘default, login’ functions of sp_msgadmin do not allow you to register Adaptive Server logins, or to create default Adaptive Server logins if the endpoint specified is a Queue Manager. See sp_msgadmin on page 48 for more information about sp_msgadmin.

Installing MQ client on Adaptive Server host machines

You must install the MQ client software on the Adaptive Server host machine.

Installing MQ client libraries

Adaptive Server dynamically loads the MQ client shared libraries. Table 2-1 shows where to install the shared libraries.
MQ authorizations

**Table 2-1: MQ client shared libraries and directories**

<table>
<thead>
<tr>
<th>Platform</th>
<th>Directory</th>
<th>Library name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Solaris 32</td>
<td>/opt/mqm/lib</td>
<td>libmqmcs.so, libmqic.so</td>
</tr>
<tr>
<td>Solaris 64</td>
<td>/opt/mqm/lib64</td>
<td>libmqmcs.so, libmqic.so</td>
</tr>
<tr>
<td>Linux 32</td>
<td>/opt/mqm/lib</td>
<td>libmqic.so</td>
</tr>
<tr>
<td>HP 64</td>
<td>/opt/mqm/lib64</td>
<td>libmqic.sl</td>
</tr>
<tr>
<td>AIX 64</td>
<td>/usr/mqm/lib64</td>
<td>libmqic_r.so(mqic_r.o)</td>
</tr>
</tbody>
</table>

You must also set the shared library search environment variable before you restart Adaptive Server. Table 2-2 lists the variables and the library names.

**Table 2-2: Shared library environment variables**

<table>
<thead>
<tr>
<th>Platform</th>
<th>Directory</th>
<th>Library name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Solaris 32</td>
<td>LD_LIBRARY_PATH</td>
<td>/opt/mqm/lib</td>
</tr>
<tr>
<td>Solaris 64</td>
<td>LD_LIBRARY_PATH</td>
<td>/opt/mqm/lib64</td>
</tr>
<tr>
<td>Linux 32</td>
<td>LD_LIBRARY_PATH</td>
<td>/opt/mqm/lib</td>
</tr>
<tr>
<td>HP 64</td>
<td>SHLIB_PATH or LD_LIBRARY_PATH</td>
<td>/opt/mqm/lib64</td>
</tr>
<tr>
<td>AIX 64</td>
<td>LIBPATH</td>
<td>/usr/mqm/lib64</td>
</tr>
</tbody>
</table>

**MQ authorizations**

MQ configuration requires the following UNIX user account and user group (principle/group) authorizations:

**Table 2-3: MQ principle/groups and their authorizations**

<table>
<thead>
<tr>
<th>MQ principle/group</th>
<th>MQ authorization</th>
</tr>
</thead>
<tbody>
<tr>
<td>OS login that is running the data server executable</td>
<td>connect, altusr, inq, and setid on Queue Manager</td>
</tr>
<tr>
<td>OS login of Adaptive Server login that is executing any messaging operation</td>
<td>inq on queue</td>
</tr>
<tr>
<td>OS login of Adaptive Server login that is executing the messaging read operation</td>
<td>get on queue</td>
</tr>
<tr>
<td>OS login of Adaptive Server login that is executing the messaging browse operation</td>
<td>browse on queue</td>
</tr>
<tr>
<td>OS login of Adaptive Server login that is executing the messaging send operation</td>
<td>put on queue</td>
</tr>
<tr>
<td>OS login of Adaptive Server login dynamic queue specified as the replyToQueueCRT, dlt on Queue Manager, and get, inq on Model Queue</td>
<td></td>
</tr>
</tbody>
</table>
 CHAPTER 3

Configuring Real Time Messaging Services

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

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Configuring RTMS</td>
<td>27</td>
</tr>
<tr>
<td>Configuring Adaptive Server for MQ</td>
<td>28</td>
</tr>
</tbody>
</table>

Configuring RTMS

To install Sybase RTMS, simply install Adaptive Server, including an RTMS requires an ASE_MESSAGING license. Follow the instructions in the installation guide for your platform. You must install Adaptive Server before you can configure RTMS.

❖ Configuring RTMS

1. Install the RTMS 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 RTMS:

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

   **Note** LD_LIBRARY_PATH must be correct for this step to succeed.

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   If you are using MQ, configure Q engines—which are Adaptive Server engines that perform only MQ client API calls—and restart:
    sp_configure "max online engines", 11
    sp_configure "max online Q engines", 1
    sp_configure "number of Q engines at startup", 1

   If the existing max online engines parameter is set to 10, this example sets it to 11 to allow one Q engine.

Configuring Adaptive Server for MQ

A Q engine uses the same amount of memory resources that Adaptive Server engines use. This is a requirement for MQ; messaging operations fail if you do not have enough Q engines. You cannot run any Adaptive Server sessions on the Q engine.

Q engines appear in sysengines, with a "_q" appended to their status:

- online_q – engine is online.
- offline_q – engine is offline.
- dormant_q – engine is dormant.

To bring a Q engine online, use the sp_engine stored procedure; an existing sp_engine works on Q engines. Use sp_configure "max online Q engines" to specify the maximum number of engines online.

Using sp_configure to configure the Q engine

You can configure the Q engine using sp_configure and the parameters discussed in this section. For more information about using sp_configure, see Adaptive Server Enterprise Reference Manual: Procedures.
max online Q engines

You can use the max online Q engines parameter with sp_configure to control the maximum the number of Adaptive Server Q engines. For example:

```
sp_configure "max online Q engines", 4
```

Valid values are:

- Minimum value: 0.
- Default value: 0.
- Maximum value: Depends on the settings using sp_configure “max online engines”.

The restrictions are:

- max online Q engines cannot be greater than max online engines minus number of engines at startup.
- The command fails if there is already an engine group referencing an engine in the range max online engines minus max online Q engines to max online engines minus 1.

For instance, if max online engines is 10, you then attempt to set max online Q engines to 4, an error is returned if there is an engine group bound to engines 6, 7, 8, or 9.

- max online Q engines cannot be greater than max online engines.

Setting max online Q engines reserves the high range of max online engines for Q engines. Once you set max online Q engines, Adaptive Server engines cannot use the engines in the range that is reserved for Q engines. For example, if you set max online engines to 10, and set max online Q engines to 4, Adaptive Server cannot use engines 6, 7, 8, and 9, and subsequent attempts to change number of engines at startup to 7, 8, or 9 fail, as does attempts to add engines 6, 7, 8, or 9 to an engine group.

Because setting max online Q engines can affect existing production environments, Sybase recommends that you increase max online engines by the same value as you set max online Q engines. For example, to set max online Q engines to 4, increase max online engines by 4 also.

number of Q engines at startup

This is an integer option that controls the number of Adaptive Server Q engines that are automatically started when Adaptive Server starts. For example:
sp_configure "number of Q engines at startup", 4

Valid values are:
• Minimum value is 0.
• Default value is 0.
• Maximum value must be less than max online Q engines.

**max native threads per engine**

A Q engine uses operating-system native threads. The max native threads per engine configuration parameter controls the maximum number of native threads that a Q engine uses. In this example, the procedure limits every Q engine to a maximum of 100 native threads:

```
sp_configure 'max native threads per engine', 100
```

The parameter has the following values:
• Minimum value is 50
• Maximum value is 1000
• Default value is 50

If there are more messaging sessions than there are native threads configured, the messaging operation blocks and waits until a native thread is released.

**Online engines and number of CPUs**

The total number of online database management systems and Q engines cannot be greater than the number of CPUs on the system.

You cannot use mgsend and msgrecv if the values of max online Q engines or number of Q engines at startup are 0.
This chapter describes a stored procedure, `sp_msgadmin` and its options, which you can use to manage and administer Real Time Messaging Services (RTMS).

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Message-related global variables</td>
<td>32</td>
</tr>
<tr>
<td><code>&lt;msgheader&gt;</code> and <code>&lt;msgproperties&gt;</code> documents</td>
<td>38</td>
</tr>
<tr>
<td>Adaptive Server-specific message properties</td>
<td>40</td>
</tr>
<tr>
<td>Keywords</td>
<td>41</td>
</tr>
<tr>
<td>Stored procedures</td>
<td>42</td>
</tr>
<tr>
<td>Built-in functions</td>
<td>42</td>
</tr>
<tr>
<td>Syntax segments</td>
<td>43</td>
</tr>
<tr>
<td><code>sp_engine</code></td>
<td>44</td>
</tr>
<tr>
<td><code>sp_msgadmin</code></td>
<td>48</td>
</tr>
<tr>
<td><code>msgconsume</code></td>
<td>57</td>
</tr>
<tr>
<td><code>msgpropcount</code></td>
<td>60</td>
</tr>
<tr>
<td><code>msgproplist</code></td>
<td>61</td>
</tr>
<tr>
<td><code>msgproppname</code></td>
<td>63</td>
</tr>
<tr>
<td><code>msgproptype</code></td>
<td>64</td>
</tr>
<tr>
<td><code>msgpropvalue</code></td>
<td>67</td>
</tr>
<tr>
<td><code>msgpublish</code></td>
<td>69</td>
</tr>
<tr>
<td><code>msgrecv</code></td>
<td>73</td>
</tr>
<tr>
<td><code>msgsend</code></td>
<td>88</td>
</tr>
<tr>
<td><code>msgsubscribe</code></td>
<td>124</td>
</tr>
<tr>
<td><code>msgunsubscribe</code></td>
<td>127</td>
</tr>
<tr>
<td><code>endpoint</code></td>
<td>130</td>
</tr>
<tr>
<td><code>option_string</code></td>
<td>133</td>
</tr>
<tr>
<td><code>sizespec</code></td>
<td>134</td>
</tr>
<tr>
<td><code>timespec</code></td>
<td>135</td>
</tr>
</tbody>
</table>
Message-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.

- MQSeries – MQ does not verify whether @@msgcorrelation consists of printable characters. Application programs should not rely on @@msgcorrelation being in the current server character set, and should only use @@msgcorrelation as a selector for subsequent messages. If @@msgcorrelation is to be returned to the application, convert it to a varbinary datatype.

- TIBCO JMS – @@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 38.

Functions that set @@msgheader include msgrecv and msgconsume.

For fields and descriptions about @@msgheader, see Table 4-1 on page 32 for MQSeries and Table 4-2 for JMS.

### Table 4-1: MQSeries @@msgheader fields and descriptions

<table>
<thead>
<tr>
<th>Property name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ApplIdentityData</td>
<td>Application data relating to identity.</td>
</tr>
<tr>
<td>ApplOriginData</td>
<td>Application data relating to origin.</td>
</tr>
<tr>
<td>CodedCharSetId</td>
<td>Numeric coded character set identifier.</td>
</tr>
<tr>
<td>CorrelId</td>
<td>Correlation identifier.</td>
</tr>
<tr>
<td>DecimalEncoding</td>
<td>This is the encoding for decimal numbers in the message payload, and is a synthesized property derived from the Encoding field. If:</td>
</tr>
<tr>
<td></td>
<td>• BigEndian – decimal numbers are big-endian</td>
</tr>
<tr>
<td></td>
<td>• LittleEndian – decimal numbers are little-endian</td>
</tr>
<tr>
<td></td>
<td>• Undefined – decimal numbers are not defined as either big-endian or little-endian</td>
</tr>
<tr>
<td>Feedback</td>
<td>Feedback status</td>
</tr>
</tbody>
</table>
### Property name | Description
--- | ---
FloatEncoding | This is the encoding for floating point numbers in the payload, and is a synthesized property derived from the Encoding field. If:
- BigEndian – floating point numbers are big-endian
- LittleEndian – floating point numbers are little-endian
- Undefined – floating point numbers are not defined as either big-endian or little-endian

Format | Format name of message data, this can be an MQ-defined format name or an application-defined format name.

GroupId | Group identifier

IntegerEncoding | This is the encoding for integers in the payload, and is a synthesized property that is derived from the Encoding field. If:
- BigEndian – integers are big endian
- LittleEndian – integers are little endian
- Undefined – the endianess of integers is undefined

LastMsgInGroup | If:
- true – message is the last message of a group
- false – message is not the last message of a group

MsgId | Message identifier

MsgInGroup | If:
- true – message is part of a group
- false – message is not part of a group

MsgSeqNumber | Message sequence number

MessageType | Message type in the form of a decimal number, unless:
- request – the message is a request message
- reply – the message is a reply message
- datagram – the message is a datagram message
- report – the message is a report message

NegativeActionNotification | This is a synthesized property, derived from the Report field. The receiving application should generate a negative-action notification (NAN) report.
- yes – receiving application should generate a NAN report message, and send it to the destinations specified in the ReplyToQ and ReplyToQMgr fields.
- no – receiving application should not generate a NAN report message.

Persistence | The persistence of the message.
If:
- persistent – the message is a persistent message
- non-persistent – the message is a non-persistent message
**Message-related global variables**

<table>
<thead>
<tr>
<th>Property name</th>
<th>Description</th>
</tr>
</thead>
</table>
| PositiveActionNotification | This is a synthesized property derived from the Report field. The receiving application should generate a Positive Action Notification (PAN) report. If:  
  - yes – receiving application should generate a PAN report message, and send it to the destinations specified in the ReplyToQ and ReplyToQMgr fields.  
  - no – receiving application should not generate a PAN report message.                                                                 |
| PutAppName              | Name of application that put the message                                                                                                                                                           |
| PutAppType              | Type of application that put the message                                                                                                                                                           |
| PutDate                 | Date when message was put                                                                                                                                                                           |
| PutTime                 | Time when message was put                                                                                                                                                                           |
| ReplyCorrelationId      | A synthesized property, derived from the Report field. Denotes what to use as the correlation ID of the report message.  
  - msgId – the correlation ID of the report message should be set to the message ID of the received message.  
  - correlationId – the correlation ID of the report message should be set to the correlation ID of the received message.                               |
| ReplyMsgId              | A synthesized property, derived from the Report field. Denotes what to use as the message ID of the report message.  
  - new – a new message ID should be used as the message ID of the report message.  
  - original – the message ID of the message received should be used as the message ID of the report message.                                        |
| ReplyToQ                | Name of reply queue                                                                                                                                                                                  |
| ReplyToQMgr             | Name of the reply queue manager                                                                                                                                                                      |
| Report                  | Report options from the message                                                                                                                                                                      |
|                         | This is a bitmap of MQRO * flags.                                                                                                                                                                    |
| UserIdentifier          | User identifier                                                                                                                                                                                        |

**Table 4-2: JMS @@msgheader fields and descriptions**

<table>
<thead>
<tr>
<th>Property name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>correlation</td>
<td>Correlation ID from the message</td>
</tr>
<tr>
<td>destination</td>
<td>The name of the destination from the message</td>
</tr>
<tr>
<td>encoding</td>
<td>The encoding name of the message</td>
</tr>
<tr>
<td>messageid</td>
<td>The message ID from the message</td>
</tr>
<tr>
<td>mode</td>
<td>Delivery mode of the message. Values:</td>
</tr>
<tr>
<td></td>
<td>- persistent</td>
</tr>
<tr>
<td></td>
<td>- nonpersistent</td>
</tr>
<tr>
<td>priority</td>
<td>The message priority</td>
</tr>
<tr>
<td>redelivered</td>
<td>The redelivery status from the message</td>
</tr>
<tr>
<td>replyto</td>
<td>The replyto name from the message</td>
</tr>
</tbody>
</table>
Contains the ID of the most recent message sent or received.

MQ Series – MQ does not verify that the @@msgid consists of printable characters. Application programs should not rely on @@msgid being in the current server character set, and should only use @@msgid as a selector for subsequent messages. If @@msgid is returning to the application, it should be converted to a varbinary datatype.

Functions that set the variable are:

- JMS – msgsend, msgpublish, msgrecv, msgconsume.
- MQ Series – msgsend, msgrecv.

@@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 38.

- JMS – the @@msgproperties are the user properties from the message.
- MQ Series – if:
  
  - The message contains one or more MQRF headers, the name-value pairs in the MQRF headers and inserted into @@msgproperties.
  
  - Since the name-value pairs in the MQRF header can have non-unique names, the names are made unique by appending a “_ddd”, where ddd is an integer extension for uniqueness.

For instance, a MQRF header with these topics:

```
MQPSTopic */baseball
MQPSTopic */baseball/world series
MQPSTopic */sports
```

Results in these properties in @@msgproperties:

```
MQPSTopic */baseball
MQPSTopic_1 */baseball/world series
MQPSTopic_2 */sports
```

Functions that set the variable are:
**Message-related global variables**

- **JMS** – `msgrecv`, `msgconsume`
- **MQ Series** – `msgrecv`

The list below lists RFH name-value pairs that are extracted from the RF header if they are present.

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>@@msgreplyqmgr</code></td>
<td>MQSeries only – contains the <code>ReplyToQmgr</code> name of the last message read.</td>
</tr>
<tr>
<td><code>@@msgreplytoinfo</code></td>
<td>Contains the name (<code>provider_url</code>, <code>queue_name</code>, <code>topic_name</code>, <code>user_name</code>) of the topic or queue name used to receive the next message. Can be a permanent or temporary destination.</td>
</tr>
</tbody>
</table>

Functions that set the variable are:
- **JMS** – `msgsend`, `msgpublish`.
- **MQSeries** – `msgsend`.

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>@@msgschem</code></td>
<td>TIBCO JMS only – contains the schema of the message or a null value. Contains the value of the Adaptive Server property <code>ase_message_body_schema</code>. For more information, see the description of the schema option in <code>msgsend</code> and <code>msgpublish</code>.</td>
</tr>
</tbody>
</table>

Functions that set the variable are: `msgsend`, `msgpublish`.

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>@@msgstatus</code></td>
<td>Contains either the integer error code of the service provider exception, or zero, if the last operation did not raise an exception.</td>
</tr>
</tbody>
</table>

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

Unrecognized names are ignored. If the value is quoted (") in the RF header, the surrounding quotes are removed. In a quoted value, if there are escaped quotes (""") within the value, doubled quotes are replaced by a single quote.

MQPSCommand  MQPSCompCode  MQPSCorrelId  MQPSDelOpts  MQPSErrorId  MQPSErrPos  MQPSSeqNum  MQPSUserld
MQPSPubOpts  MQPSPubTime  MQPSQMGrName  MQPSSubIdentity  MQPSReason  MQPSReasonText  MQPSSubUserData  MQPSUserld
MQPSStreamName  MQPStringData  MQPSSubName  MQPSSubUserData  MQPSTopic  MQPSTopic

@@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.

MQ Series – contains provider error message of last messaging operation. The MQ client libraries do not provide localized error messages, so you see an error message such as:

MQ API call failed with reason code '%s' (%d)

The “%s” is substituted with the MQ mnemonic for the MQ reason code.
The “%d” is substituted with the decimal MQ reason code.

Functions that set the variable are:
- JMS – msgsend, msgpublish, msgrecv, msgconsume.
- MQ Series – msgsend, msgrecv.

@@msgtimestamp
Contains the timestamp included in the message last sent.

Functions that set the variable are: msgsend, msgpublish.

Examples
MQSeries only – shows request/reply messaging using both @@msgreplytoinfo and @@msgcorrelation:

Session 1 (requester)
```
select msgsend('sender message',
   'ibm_mq:channel1/TCP/host1(5678)' + '?qmgr=QM1' + ',queue=Q100',
   option 'msgType=request',
   message property
   'correlationId=0x123456'
   + 'replyToQueue=Q200')
```

Session 2 (receiver)
```
select msgrecv(
   'ibm_mq:channel1/TCP/host1(5678)' + '?qmgr=QM1' + ',queue=Q100')
```

```
select msgsend('receiver reply',
   @@msgreplytoinfo,
   option 'msgType=reply'
   message property
   'correlationId=' + @@msgcorrelation)
```
In this example:

1. Session 1 sends the request message to Q100, and expects the reply message on Q200. It sets the correlation to 0x123456.

2. Session 2 reads a message from Q100, sends a reply message to Q200, and specifies the correlation to 0x123456. The reply queue is obtained from the message that was just read.

3. Session 1 reads the reply message from Q200, wanting only message with correlation 0x123456.

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 88:

  JMS only – 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**

```xml
<!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:

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

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

select rtrim (@@msgproperties)

<?xml version='1.0' encoding='UTF-8' standalone='yes' ?>
<msgproperties 
RTMS_MSGBODY_FORMAT=''string''
ASE_RTMS_CHARSET='1'
ASE_RTMS_VERSION=''1.0''
ASE_VERSION=''12.5.0.0''
shape=''square''
color=''red'' > 
</msgproperties>

select rtrim (@@msgheader)

<?xml version='1.0' encoding='UTF-8' standalone='yes' ?>
<msgheader 
type=''null''
timestamp='1080092021000'
replyto=''queue.sample''
redelivered='false'
priority='4'
messageid=''ID:E4JMS-SERVER.73018656B39:1''
ttl='0'
destination=''queue.sample''
mode='2'
correlation=''null''
encoding=''null'' > 
```
Adaptive Server-specific message properties

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

TIBCO JMS – to help with debugging, monitoring, and so forth, predefined properties specific to Adaptive Server are included in the properties portion of the TIBCO JMS 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 4-3 describes these message properties.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>When to use</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASE_RTMS_CHARSET</td>
<td>Character set encoding of sent data.</td>
<td>Always</td>
</tr>
<tr>
<td>ASE_MSGBODY_SCHEMA</td>
<td>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 <code>msgsend</code>. If <code>ASE_MSGBODY_FORMAT</code> is <code>xml</code>, this property contains the XML schema describing the payload. This schema is not truncated, even if its value exceeds 16K.</td>
<td>Always</td>
</tr>
<tr>
<td>ASE_MSGBODY_FORMAT</td>
<td>The format of the message body: <code>xml</code>, <code>string</code> (in server character set), <code>binary</code>, and <code>unicode</code> (unichar in network order).</td>
<td>Always</td>
</tr>
</tbody>
</table>
### Table 4-4: Double and triple keywords in RTMS

<table>
<thead>
<tr>
<th>JMS or MQ Series</th>
<th>Keywords</th>
<th>Legal commands and functions using keywords</th>
</tr>
</thead>
<tbody>
<tr>
<td>Both</td>
<td>message header</td>
<td>select msgsend(,,message header,,)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>select msgpublish(,,message header,,)</td>
</tr>
<tr>
<td>Both</td>
<td>message property</td>
<td>select msgsend(,,message property,,)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>select msgpublish(,,message property,,)</td>
</tr>
<tr>
<td>JMS</td>
<td>message selector</td>
<td>select msgrecv(,,message selector,,)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>select msgconsume(,,message selector,,)</td>
</tr>
<tr>
<td>JMS</td>
<td>with retain</td>
<td>select msgunsubscribe(,,with retain,,)</td>
</tr>
<tr>
<td>JMS</td>
<td>with remove</td>
<td>select msgunsubscribe(,,with remove,,)</td>
</tr>
<tr>
<td>Both</td>
<td>transactional messaging none</td>
<td>set transactional messaging none</td>
</tr>
<tr>
<td>Both</td>
<td>transactional messaging simple</td>
<td>set transactional messaging simple</td>
</tr>
<tr>
<td>Both</td>
<td>transactional messaging full</td>
<td>set transactional messaging full</td>
</tr>
</tbody>
</table>

### Keywords

Table 4-4 shows the keywords specific to RTMS, and the functions in which these keywords can be legally used.

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

The two stored procedures you use with this feature are:

- `sp_msgadmin` on page 48
- `sp_engine` on page 44

`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.

**Note** `sp_addexeclass` does not accept MQSeries Q engines for the `anyengine` and `lastonline` parameters.

Built-in functions

The section in this chapter on built-in functions describes the SQL functions for administering Real Time Messaging, and the general format of option strings. See Table 4-3 on page 40 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 57
- `msgpropcount` on page 60
- `msgproplist` on page 61
- `msgpropname` on page 63
- `msgproptype` on page 64
- `msgpropvalue` on page 67
- `msgpublish` on page 69
• msgrecv on page 73
• msgsend on page 88
• msgsubscribe on page 124

Syntax segments

The section in this chapter on syntax segments describes the portions of SQL syntax and constraints used in administering Real Time Messaging.

The syntax segments listed in this chapter, and their page numbers, are:
• endpoint on page 130
• option_string on page 133
• sizespec on page 134
• timespec on page 135
sp_engine

Description
Enables you to bring a Q engine online or take it offline.

Syntax
sp_engine "online | offline | can_offline | shutdown
               | q_online | q_offline | q_can_offline | q_shutdown" , [ engine_id]

Parameters

can_offline
returns information on whether an engine can be brought offline. If the
engine cannot be brought offline, you see the spids of the Adaptive
Server sessions that prevent the engine from being offline. You cannot
use this to specify a Q engine.

engine_id
the ID of the engine

The type of the engine that is you specify must match the command
(online, q_online, and so on). For example, you cannot specify a non-Q
engine with q_offline, and you cannot specify a Q engine with offline.

This parameter is required for offline, q_offline, can_offline,
q_can_offline, shutdown and q_shutdown.

This parameter is not required for online, q_online.

online
brings an engine online. The value of sp_configure "max online Q
engines" must be greater than the current number of Q engines online.
You must use quotes because "online" is a reserved keyword. You
cannot use this to specify a Q engine.

offline
brings an engine offline. You can also use engine_id to specify an engine
to bring offline. You cannot use this to specify a Q engine.

q_can_offline
returns information on whether a Q engine can be brought offline. If the
engine cannot be brought offline, you see the spids of the Adaptive
Server sessions that prevent the engine from being offline. You must
use engine_id to specify whether a Q engine can be taken offline.

q_offline
brings a Q engine offline. You must use engine_id to specify an engine
to bring offline.

q_online
brings the next Q engine online.
**q_shutdown**
forces an engine offline. If there are any tasks with an affinity to this engine, they are killed after a five-minute wait. You must use quotes, as shutdown is a reserved keyword. You must use `engine_id` to specify an whether the Q engine can shut down.

**shutdown**
forces an engine offline. If there are any tasks with an affinity to this engine, they are killed after a five-minute wait. You must use quotes, as shutdown is a reserved keyword. You cannot use this to specify a Q engine.

**Examples**

**Example 1** Manually brings a Q engine online. are platform specific

```
sp_engine 'q_online'
go
```

```
02:00000:00000:2005/06/08 12:52:21.24 kernel Q engine 2, os pid 20025 online
```

**Example 2** Takes a Q engine offline:

```
1> select engine, status from sysengines
2> go
```

```
engine status
------ ------------
0 online
1 online_q
2 online_q
(3 rows affected)
```

```
1> sp_engine 'q_offline', 1
2> go
```

```
00:00000:00000:2005/06/08 12:55:54.25 kernel engine 2, os pid 20025 offline
```

```
1> select engine, status from sysengines
```
2> go
    engine status
    ------    --------
    0 online
    1 online_q
    (2 rows affected)

**Example 3** Checks to see you can take a Q engine offline:

1> select engine, status from sysengines
2> go
    engine status
    ------    --------
    0 online
    1 online_q
    (2 rows affected)
1> sp_engine 'q_can_offline', 1
2> go
    spid: 13 has outstanding rtms-connection connections.

**Example 4** Shuts down a Q engine:

1> select engine, status from sysengines
2> go
    engine status
    ------    --------
    0 online
    1 online_q
    (2 rows affected)
1> sp_engine 'q_shutdown', 1
2> go
    (return status = 0)
1> select engine, status from sysengines
2> go
    engine status
    ------    --------
    0 online
    (1 row affected)

**Usage**

- online, offline, can_offline, and shutdown affect only non-Q engines. You see an error if you specify a Q engine with these parameters.
• q_online, q_offline, q_can_offline, and q_shutdown affect only Q engines. You see an error if you specify a non-Q engine using these parameters.

• You cannot shut down or take engine 0 offline.

• You can determine the status of an engine, and which engines are currently online with the following query:

  ```sql
  select engine, status from sysengines
  where status = "online"
  ```

• online and shutdown are keywords and must be enclosed in quotes.

• You can bring engines online only if max online Q engines is greater than the current number of engines with an online status, and if enough CPU is available to support the additional engine.

• An engine offline can fail or might not immediately take effect if there are server processes with an affinity to that engine.

Permissions

You must be a System Administrator to bring engines online or take them offline.
**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]
  | 'subscription', [ subscription_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]]]
  | ['publisher', publisher_name, endpoint, topic
    [, broker_queue[, request_queue[, options]]]]
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.

**Note** You cannot use `sp_msgadmin 'default', 'login'` if `endpoint` is an MQ Queue Manager.

---

**delivery_option**

specifies 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*.
**sp_msgadmin**

**endpoint**

is the topic to which the subscription is addressed. See the description of *endpoint* in msgsend on page 88.

**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.

**Note** You cannot use `sp_msgadmin 'register', 'login'` if *endpoint* is an MQ Queue Manager.

- 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 values are:

- TIBCO_JMS
- IBM_MQ

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.

Note You cannot use sp_msgadmin 'register', 'login' or sp_msgadmin 'register', 'subscription' if endpoint is an MQ Queue Manager.

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 73.
**sp_msgadmin**

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

- register – registers a subscription.

**Note** You cannot use `sp_msgadmin 'register' 'subscription'` if the endpoint is an MQ Queue Manager.

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

  \[
  \text{sp_msgadmin 'register' 'subscription', subscription_name, ...}
  \]

  \[
  \text{subscription_name}
  \]

  is a subscription name.

**Examples**

**Example 1** MQSeries – registers the "mq_provider_1" messaging provider, which has a class of IBM_MQ and a URL of chanl1/TCP/host1(5678):

  \[
  \text{sp_msgadmin 'register', 'provider', 'mq_provider_1', 'ibm_mq',}
  \text{'chanl1/TCP/host1(5678)'}
  \]

**Example 2** TIBCO JMS – specifies the default login that applies to all unmapped Adaptive Server logins, when using a specified messaging provider for either sending or receiving:

  \[
  \text{sp_msgadmin 'default', 'login', 'my_jms_provider',}
  \text{'jms_user1', 'jms_user1_password'}
  \]

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

**Example 3** TIBCO JMS – specifies the default login:

  \[
  \text{sp_msgadmin 'default', 'login', 'one_jms_provider',}
  \text{'loginsa', 'abcdef123456'}
  \]

**Example 4** TIBCO JMS – describes the syntax for `sp_msgadmin 'list'`:

  \[
  \text{sp_msgadmin 'help', 'list'}
  \]

**Example 5** TIBCO JMS – checks the default login:

  \[
  \text{sp_msgadmin 'list', 'login', 'my_jms_provider'}
  \]
Example 6  TIBCO JMS – lists the details for the user with a login of “loginsa”:

```sql
sp_msgadmin 'list', 'login', 'my.jms_provider', 'loginsa'
```

Example 7  TIBCO JMS – lists the details for the “my.jms_provider” message provider:

```sql
sp_msgadmin 'list', 'provider', 'my.jms_provider'
```

Example 8  TIBCO JMS – lists the details for subscription “subscription_1”:

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

Example 9  TIBCO JMS – registers the login “ase_login1”, using messaging provider login “jms_user1”, and messaging provider name “my.jms_provider”:

```sql
sp_msgadmin 'register', 'login', 'my.jms_provider', 'ase_login1', 'jms_user1', 'jms_user1_password'
```

Example 10  TIBCO JMS – registers a login using the messaging provider login “jms_user1”, and a specified password used for all unmapped Adaptive Server logins:

```sql
sp_msgadmin 'register', 'login', 'my.jms_provider', null, 'jms_user1', 'jms_user1_password'
```

Example 11  TIBCO JMS – registers a login with the messaging provider login “jms_user1”, and a specified password used for all Adaptive Server logins that have sa_role permissions:

```sql
sp_msgadmin 'register', 'login', 'my.jms_provider', null, 'jms_user1', 'jms_user1_password', 'sa_role'
```

Example 12  TIBCO JMS – registers the “my.jms_provider” messaging provider, which has a class of TIBCO_JMS and an IP of 10.23.233.32:4823 as its address:

```sql
sp_msgadmin 'register', 'provider', 'my.jms_provider', 'TIBCO_JMS', 'tcp://10.23.233.32:4823'
```

Example 13  TIBCO JMS – registers a durable subscription named “durable_sub1”, then `sp_msgadmin 'list' displays information about the new subscription.

```sql
sp_msgadmin
```
sp_msgadmin

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

Example 14 TIBCO JMS – registers “subscription_1”, a nondurable subscription.
sp_msgadmin 'register', 'subscription', 'subscription_1',
'my_jms_provider?topic=topic.sample'

Note You must first use sp_msgadmin register, provider
to register “my_jms_provider”.

Example 15 TIBCO JMS – removes the default login:
sp_msgadmin 'remove', 'login', 'my_jms_provider'

Example 16 TIBCO JMS – removes the messaging provider “my_jms_provider”:
sp_msgadmin 'remove', 'provider', 'my_jms_provider'

Example 17 TIBCO JMS – removes the Adaptive Server login “ase_login1” associated with the messaging provider “my_jms_provider”:
sp_msgadmin 'remove', 'login', 'my_jms_provider',
'ase_login1'

Example 18 TIBCO JMS – removes the default login, indicated by a null login parameter:
sp_msgadmin 'remove', 'login', 'my_jms_provider',
null

Example 19 TIBCO JMS – removes all logins for role sa_role on “my_jms_provider”:
sp_msgadmin 'remove', 'login', 'my_jms_provider',
null, 'sa_role'

Example 20 TIBCO JMS – 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. Explicit 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.

- MQSeries only – if you enter an endpoint using a registered provider, using msgsubscribe, msgunsubscribe, msgpublish, and msgconsume return errors.

- See sp_msgadmin on page 48 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_msgadmin 'register'
- sp_msgadmin 'remove'
Any user can issue:

- `sp_msgadmin 'help'`
- `sp_msgadmin 'list'`
msgconsume

Description
TIBCO JMS only – 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 133. The special options to use when consuming a message are described in Table 4-5:

<table>
<thead>
<tr>
<th>option values</th>
<th>option_string values</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>timeout</td>
<td>timespec</td>
<td>-1</td>
<td>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. timeout uses the timespec option. See timespec on page 135 for more information.</td>
</tr>
<tr>
<td>requeue</td>
<td>string</td>
<td>None</td>
<td>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.</td>
</tr>
</tbody>
</table>

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

returns
specifies the clause that you want returned.
**msgconsume**

**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`
- `univarchar(n)`

**Examples**

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

```sql
sp_msgadmin 'register', 'subscription', 'subscription_1',
    'my_jms_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:

```sql
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:

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

Forwards a message:

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

Reads a message and returns it as a `varbinary`:

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

- Unrecognized option names result in an error.

**Note** This behavior is new with version 12.5.3a, and differs from previous versions.

- `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 messaging 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 “Message-related global variables” on page 32.

  - 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**
```plaintext
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 `@@msgproperties`.
- `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
<?xml version='1.0' encoding='UTF-8' standalone='yes' ?>
<msgproperties
  trade_name='Acme Maintenance ("Quick & Safe")'>
</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:
```sql
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**
```sql
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 `@@msgproperties` 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
<?xml version='1.0' encoding='UTF-8' standalone='yes' ?>
<msgproperties
    trade_name='Acme Maintenance ("Quick & Safe")'>
</msgproperties>
```

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

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

```sql
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:

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

  However, this one is not:

  ```sql
  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**
```sql
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 `@@msgproperties` 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
<?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:
```sql
select msgpropname(8, @@msgproperties)
```

**Example 2** Returns a null value, because the ninth property does not exist:
```sql
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 ])
                  prop_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 `@@msgprproperties` 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
```
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)

Usage

- MQSeries – when you use `msgproptype` to query one of the following binary fields contained in the MQ message header, the string “Hex” is returned:
  - MsgId
  - CorrelId
  - GroupId
  - Encoding
For example, the following returns “Hex”:

```sql
select msgproptype ('Encoding', @msgheader)
```
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 `@@msgproperties` 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
<?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(@@msgproperties, 'trade_name')
---------------
('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:

```sql
select msgpropvalue (msgpropname(8, @@msgproperties))
```
**msgpublish**

**Description**
TIBCO JMS only – 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 133.

  The options you can specify for msgsend are in Table 4-6 on page 71.

- **properties_clause**
  is either an option_string or one of the options listed in the following tables. The options described in Table 4-6 on page 71 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 4-7 on page 71, it is set as a property in the message properties of the message sent.
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.jms_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 4-6 lists the options you can specify for msgsend for TIBCO JMS.

**Table 4-6: option_string values for msgpublish**

<table>
<thead>
<tr>
<th>Option</th>
<th>Values</th>
<th>Default</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>schema</td>
<td>• no</td>
<td>no</td>
<td>Enter one of these values:</td>
</tr>
<tr>
<td></td>
<td>• yes</td>
<td></td>
<td>• user_schema – is a user-supplied schema describing the message_body.</td>
</tr>
<tr>
<td></td>
<td>• &quot;user_schema&quot;</td>
<td></td>
<td>• no – indicates that no schema is generated and sent out as part of the</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>message.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• yes – indicates that Adaptive Server generates an XML schema for the</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>message. yes is meaningful only in a message_body that uses the</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>select_for_xml parameter. select_for_xml generates a SQLX-formatted</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>representation of the SQL result set. The generated XML schema is a</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>SQLX-formatted schema that describes the result set document.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>The schema is included in the message as ASE_MSGBODY_SCHEMA property.</td>
</tr>
<tr>
<td>type</td>
<td>text or bytes</td>
<td>text</td>
<td>The message type to send.</td>
</tr>
</tbody>
</table>

- Table 4-7 lists the options and values for the properties_clause parameter. If you use a property not listed in Table 4-7, it is set as a property in the message properties of the message sent.

**Table 4-7: Values for the msgpublish properties_clause parameter**

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

<table>
<thead>
<tr>
<th>Option</th>
<th>Values</th>
<th>Default</th>
<th>Disposition</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>priority</td>
<td>1 to 10</td>
<td>4</td>
<td>header</td>
<td>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.</td>
</tr>
<tr>
<td>replyqueue</td>
<td>A string containing a <code>queue_name</code></td>
<td>none</td>
<td>header</td>
<td>The value of <code>queue_name</code> or <code>topic_name</code> must be <code>sub_temp</code>. The type of the temporary destination, queue or topic, depends on whether you specify <code>replyqueue</code> or <code>replytopic</code>. 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.</td>
</tr>
<tr>
<td>replytopic</td>
<td>A string containing a <code>topic_name</code></td>
<td>none</td>
<td>header</td>
<td></td>
</tr>
<tr>
<td>ttl</td>
<td>$0 – (2^{53}-1)$</td>
<td>0</td>
<td>header</td>
<td><code>ttl</code> 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. <code>ttl</code> uses the <code>timespec</code> option. See <code>timespec</code> on page 135 for more information.</td>
</tr>
</tbody>
</table>

#### 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_filter_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.

Any message_filter specified to msgrecv is ignored if the provider class is “ibm_mq.”
**msgrecv**

`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 4-8 on page 77 for MQSeries, and Table 4-9 on page 84 for TIBCO JMS.

**Note** Unrecognized option names result in an error.

**option_string**

is the general format of the `option_string` is specified in option_string on page 133. The options for `msgrecv` are described in Table 4-8 on page 77 for MQSeries and Table 4-9 on page 84 for JMS.

**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`
- `univarchar(n)`

**Examples**

**Example 1** MQSeries – a message is read from the queue Q1 with a specified timeout. If no messages are available on Q1 before the timeout of 3 seconds, a null value is returned:

```sql
select msgrecv(
    'ibm_mq:chnl1/TCP/host1(5678)?qmgr=QM,queue=Q1',
    option 'timeout=3s'
)
```
Example 2 MQSeries – a correlationId is specified without a timeout. The call returns when a message matching the correlationId is available on the queue:

```sql
select msgrecv('ibm_mq:chnl1/TCP/host1(5678)?qmgr=QM,queue=Q1', option 'correlationId=x67a12z99')
```

Example 3 MQSeries – a groupId is specified, as well as allMsgsInGroup, but a timeout is not specified. This call blocks until all the messages for the groupId specified are available on the queue:

```sql
select msgrecv('ibm_mq:chnl1/TCP/host1(5678)?qmgr=QM,queue=Q1', option 'groupId=g7853b77,allMsgsInGroup=yes')
```

Example 4 MQSeries – these messages already exist on the queue:

```
AA BB CC DD EE FF GG HH
```

The first three messages are read in browse mode (AA-CC), and CC is removed. The browse cursor is then set back to the beginning, and three messages are read in browse mode (AA-DD), and DD is removed. The read that removes CC causes CC to not be included when the browse is repositioned at the beginning. Finally, a read is performed with position set to next, which reads and removes AA. When this example completes, the messages AA, CC, and DD will no longer remain on the queue.

-- Browse cursor at the beginning, this will return 'AA'
select msgrecv('ibm_mq:chnl1/TCP/host1(5678)?qmgr=QM,queue=Q1', option 'inputMode=browse+Qdefault,browse=first')

-- Browse the next message, this will return 'BB'
select msgrecv('ibm_mq:chnl1/TCP/host1(5678)?qmgr=QM,queue=Q1', option 'inputMode=browse+Qdefault,browse=next')

-- Browse the next message, this will return 'CC'
select msgrecv('ibm_mq:chnl1/TCP/host1(5678)?qmgr=QM,queue=Q1', option 'inputMode=browse+Qdefault,browse=next')

-- Remove the message under the browse cursor, this will return 'CC'
select msgrecv('ibm_mq:chnl1/TCP/host1(5678)?qmgr=QM,queue=Q1', option 'inputMode=browse+Qdefault,position=cursor')
-- Reposition browse cursor at the beginning, this will return 'AA'
select msgrecv(
     'ibm_mq:chnl1/TCP/host1(5678)?qmgr=QM,queue=Q1',
     option 'inputMode=browse+Qdefault,browse=first')

-- Browse the next message, this will return 'BB'
select msgrecv(
     'ibm_mq:chnl1/TCP/host1(5678)?qmgr=QM,queue=Q1',
     option 'inputMode=browse+Qdefault,browse=next')

-- Browse the next message, this will return 'DD'
select msgrecv(
     'ibm_mq:chnl1/TCP/host1(5678)?qmgr=QM,queue=Q1',
     option 'inputMode=browse+Qdefault,browse=next')

-- Read the message under the cursor, this will return 'DD'
select msgrecv(
     'ibm_mq:chnl1/TCP/host1(5678)?qmgr=QM,queue=Q1',
     option 'inputMode=browse+Qdefault,position=cursor')

-- Read the next message in queue order, this will return 'AA'
select msgrecv(
     'ibm_mq:chnl1/TCP/host1(5678)?qmgr=QM,queue=Q1',
     option 'inputMode=browse+Qdefault,position=next')

Example 5 TIBCO JMS – Receives a message from the specified end_point:
select msgrecv
     ('tibco_jms:tcp://my_jms_host:7222?queue=queue.sample,' +
     +'user=jms_user1,password=jms_user1_password')

Example 6 TIBCO JMS – receives a message from the specified end_point, using the timeout option and specifying a message selector:
declare @mymsg varchar (16384)
select @mymsg = msgrecv('my_jms_provider?queue=queue.sample',
option 'timeout=1000'
MESSAGE SELECTOR 'correlationID = ''MSG_001''')

Example 7 TIBCO JMS – forwards a message to the specified endpoint:
select msgsend(msgrecv('my_jms_provider?queue=queue.sample'),
   'another_jms_provider?queue=queue2')

Example 8 TIBCO JMS – this msgrecv call only consumes messages from queue.sample when the message property "Name" is equal to "John Smith":

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

**Example 9** TIBCO JMS – illustrates how to insert a text message into a table:

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

**Example 10** TIBCO JMS – this example reads a message and returns it as a varbinary.

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

Usage

- MQSeries – Table 4-8 on page 77 lists the available *option* and *option_string* values for properties of `msgrecv`.

<table>
<thead>
<tr>
<th><strong>option values</strong></th>
<th><strong>option_string values</strong></th>
<th><strong>Default</strong></th>
<th><strong>Description</strong></th>
</tr>
</thead>
</table>
| allMsgsInGroup        | *yes*                    | no          | This option is ignored unless you specify `groupid`. When you specify:
|                       | *no*                     |             | • *yes* – all logical messages of a group must be present on the queue before the first message of a group is returned.
|                       |                           |             | • *no* – not all logical messages of a group are required to be present on the queue before returning the first message of a group. |
| allSegments           | *yes*                    | no          | When you specify:
|                       | *no*                     |             | • *yes* – all messages of a segmented message must be present on the queue before the first message segment is returned.
|                       |                           |             | • *no* – not all messages of a segmented message are required to be present before returning the first message segment. |
**msgrecv**

<table>
<thead>
<tr>
<th><strong>option values</strong></th>
<th><strong>option_string values</strong></th>
<th><strong>Default</strong></th>
<th><strong>Description</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>browse</td>
<td>next</td>
<td>null</td>
<td>If you set the browse property to <code>null</code>, the message is read and removed from the queue. The position option controls which message is read. If you set the value to anything other than <code>null</code>, the message is read but not removed from the queue. The ordering depends on the default ordering of the queue (first-in, first-out or priority) If you also specify:</td>
</tr>
<tr>
<td></td>
<td>next+Lock</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>first</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>first+Lock</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>cursor</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>cursor+Lock</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>reopen</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>reopen+Lock</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>unlock</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>null</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

If you specify the following for `browse`:
- **next** – the next message is returned.
- **next+Lock** – the message is returned, and the message is locked so that other readers cannot remove it.
- **first** – the first message is returned. If you specify `browse=first` after you issue one or more `browse=next` options, the browse cursor repositions to the starting position where the queue was opened.
- **first+Lock** – the first message is returned, and the message is locked so that other readers cannot remove it.
- **cursor** – the message under the browse cursor is returned. Do not use `browse=cursor` without first performing `browse=first`, `browse=first+Lock`, `browse=next`, or `browse=next+Lock`. Repeating `browse=cursor` returns the same message.
- **cursor+Lock** – the message under the cursor is returned, and the message is locked so that other readers cannot remove it.
- **reopen** – the browse cursor is closed, reopened, and positioned at the start. For priority queues, if a higher priority message comes in since the last open, that message appears at the start of the queue.
- **reopen+Lock** – the browse cursor is closed, reopened, positioned at the start, and the first message is locked so that other readers cannot remove it.
- **unlock** – the message under the cursor is unlocked and returned.
bufferLength sizespec
0 or 1 – value

The messaging built-in function attempts to allocate a buffer of this length. The command fails if there is not enough memory to allocate the buffer.

When you specify msgrecv to return text or image, msgrecv assumes that the message size is the largest message that the specified queue can accommodate, and uses the maxMsgLength queue property. Increase messaging memory if you set maxMsgLength at:

- Its default of 4MB, or
- A value that is much larger than the actual length of the messages.

Sybase recommends you set the maxMsgLength queue property to the minimum allowed for the application so Adaptive Server can use the least amount of memory to read the message. To set maxMsgLength, use the MQSC tool to change the MAXMSGL attribute on the queue.

Defaults

bufferLength defaults to either the:
- Minimum of the maxMsgLength that is defined for the Queue Manager and the target queue, or
- The length of the return type if it is not text, image, or java.lang.String.

0 indicates to use the default.

For pub/sub messages, bufferLength must include the length of the message topics, including the MQRF header.

closeAfterRecv * yes
* no
no

If:
- yes – the queue closes after the current msgrecv operation, allowing the queue to be reopened with a different input mode on subsequent msgrecv calls.
- no – the queue remains open after the current msgrecv operation.

completeMsg * yes
* no
yes

If:
- yes – segmented messages are returned as a single message.
- no – if there are segmented messages, each segment is returned as a separate message.

completeMsg should have the same setting for all calls to msgrecv for the same endpoint.
Correlation ID of message to read. 

As selection option, you can use `correlationId` to select specific messages in your queue.

MQ defines this field as “unsigned char” that can support binary values. To enter a binary string as the `correlationId`, use “0x...” as the value. Do not add quote marks around the value.

The name of the expected message format. If specified, and the `formatName` field of the message does not match, the message is not read. See the `requeue` option in this table for more information.

MQ limits this string to 8 bytes.

Group ID of message to read. This is a selection option. MQ defines this field as “unsigned char”, which means that it can support binary values. If you want to enter a binary string as the `msgId`, use “0x...” as the value. Do not quote the value, or it is taken to be a quoted string.
### CHAPTER 4 SQL Reference

**inputMode**

- **browse**
- **Qdefault**
- **shared**
- **exclusive**
- **browse+Qdefault**
- **browse+shared**
- **browse+exclusive**

The values for `inputMode` open the MQ queue in the following ways:

- **browse** – opened for browsing only. The Queue Manager produces an error when you attempt a destructive read.
- **Qdefault** – opened in the default input mode as defined for the queue.
- **shared** – opened in shared input mode. You receive an error if the queue is already opened in exclusive mode by another MQ handle.
- **exclusive** – opened in exclusive input mode. You receive an error if the queue is already opened in shared or exclusive mode by another MQ handle.
- **browse+Qdefault** – opened for browse- and shared-input mode.
- **browse+shared** – opened for browse- and shared-input mode. You get an error if the queue is already opened in exclusive mode by another MQ handle.
- **browse+exclusive** – opened for browse- and exclusive-input mode. You get an error if the queue is already opened in shared or exclusive mode by another MQ handle.

`inputMode` is valid only for `msgrecv`.

For any endpoint, you must specify `inputMode` either:

- On the first `msgrecv` operation, or
- After you specify `closeAfterRecv`.

Attempting to change the value of `inputMode` across calls may cause unexpected results.

<table>
<thead>
<tr>
<th><code>option</code> values</th>
<th><code>option_string</code> values</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>msgId</code></td>
<td><code>null</code></td>
<td><code>null</code></td>
<td>Message ID of message to read.</td>
</tr>
<tr>
<td></td>
<td><code>string</code></td>
<td></td>
<td>As selection option, you can use <code>msgId</code> to select specific messages in your queue.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>MQ defines this field as “BYTE array” that can support binary values. To enter a binary string as the <code>msgId</code>, use “0x...” as the value. Do not add quote marks around value, as that is interpreted as a quoted string.</td>
</tr>
<tr>
<td><code>offset</code></td>
<td><code>integer</code> between -1, 0 – maxint</td>
<td></td>
<td>Offset of message to read.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>If -1, the offset is not specified.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>As selection option, you can use <code>offset</code> to select specific messages in your queue.</td>
</tr>
</tbody>
</table>
When ordering is:

- **logical** – the messages are read in logical order according to groupId, sequenceId, and offsets in the messages.
- **physical** – the messages are read in the order in which they appear on the queue.

<table>
<thead>
<tr>
<th>position values</th>
<th>option_string values</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>next</td>
<td>next</td>
<td>next</td>
<td>position controls which message is returned. Depending on what inputMode value you specify, there are one or two “read” positions:</td>
</tr>
<tr>
<td>cursor</td>
<td></td>
<td></td>
<td>- “Normal” – the default read position where destructive reads normally occur. When a queue is opened, the “normal” read position is positioned on the first message in the queue.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>- “Browse cursor” – where the read position has been positioned by a previous call where browse was specified. When a queue is opened for browse, the “browse cursor” is positioned before the first message in the queue. “Browse cursor” is used only with browse only for browse+Qdefault, browse+shared, and browse+exclusive.</td>
</tr>
</tbody>
</table>

If:

- next – the current message at the “normal” read position is returned. The “normal” read position is moved forward to the message after the message returned.
- cursor – the current message at the “browse cursor” is returned. MQ Queue Manager raises an error if the “browse cursor” has not yet been positioned. The “browse cursor” is moved forward to the message after the message returned.

The MQ Queue Manager applies the following before determining what message to return:

- The default ordering of the queue (priority or first-in, first-out)
- Any selection criteria specified (messageId, correlationId, groupId, sequenceId, or offset)
<table>
<thead>
<tr>
<th><strong>option values</strong></th>
<th><strong>option_string values</strong></th>
<th><strong>Default</strong></th>
<th><strong>Description</strong></th>
</tr>
</thead>
</table>
| requeue          | • null                  | null        | This must be a full URI of the endpoints. The read message is requeued to the queue specified if:  
|                  | • string                |             | • msgrecv reads a message when formatName is specified.  
|                  |                         |             | • The read message has a different formatName.  
|                  |                         |             | • requeue is not null.  
|                  |                         |             | If the message cannot be requeued to the queue specified, the message is left on the queue where it was read, and an exception is raised. MQ limits this string to 48 bytes. |
| sequenceId       | integer between -1, – 9,999,999 | -1          | Sequence ID of message to read. If -1, the sequence ID is not specified. As selection option, you can use sequenceId to select specific messages in your queue. |
| truncationAllowed| • yes                   | no          | You can truncate the message when:  
|                  | • no                    |             | • The buffer used to read the message (bufferLength, or length of the returned datatype).  
|                  |                         |             | • The buffer is smaller than the length of the message. Specify as:  
|                  |                         |             | • yes – to allow truncation.  
|                  |                         |             | • no – to not allow truncation. The read fails when the value is no and message is truncated. |
| timeout          | timespec between -1, 0 – (2^{32}–1) | -1          | Specifies the timeout. If:  
|                  |                         |             | • -1 – there is no timeout.  
|                  |                         |             | • timeout is specified as an integer – the value is to be taken in milliseconds. See timespec on page 135 for more information. |

- TIBCO JMS – Table 4-9 on page 84 lists the available option and option_string values for properties of msgrecv.
**msgrecv**

### Table 4-9: TIBCO JMS option and option_string values for msgrecv

<table>
<thead>
<tr>
<th><strong>option values</strong></th>
<th><strong>option_string values</strong></th>
<th><strong>Default</strong></th>
<th><strong>Description</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>requeue</td>
<td>string</td>
<td>None</td>
<td>The name of a destination, queue, or topic on which to requeue messages that Adaptive Server cannot process. If <code>requeue</code> is not specified, and the message cannot be processed, an error message appears. The endpoint specified must be on the same messaging provider as <code>msgconsume</code> and <code>msgrecv</code>.</td>
</tr>
<tr>
<td>timeout</td>
<td>timespec</td>
<td>-1</td>
<td>By default, <code>msgrecv</code> is a blocking command, which blocks the message until it reads the next message from the message bus. If <code>timeout</code> is not -1, <code>msgrecv</code> returns a null value when the <code>timeout</code> interval lapses without reading a message. The values are in numbers of milliseconds. See <code>timespec</code> on page 135 for more information.</td>
</tr>
</tbody>
</table>

- Unrecognized option names result in an error.

**Note** This behavior is new with version 12.5.3a, and differs from previous versions.

- See section “@@msgheader” on page 32 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 messaging service provider as the endpoint used in `msgrecv`.

- The message includes the binary value of the datatype according to the byte ordering of the host machine.
• 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 67.
  • The general format of <msgheader> and <msgproperties> is
    described in “Message-related global variables” on page 32.

MQSeries
These are valid only if the provider class is “ibm_mq”:
• The msgId, correlationId, groupId, sequenceId, and offset options act as
  match criteria for selecting messages. When specified, the next
  message matching the values specified are returned. The qualification
  is performed by the WebSphere MQSeries Queue Manager.
• If the MQMD.Format field of the message received is “MQSTR”, the
  data is assumed to be character data, and can be returned as text or
  varchar. Any other format name can be returned only as image or
  binary. One special case is if MQMD.Format is “MQHRF”. In this case,
  the MQRFH.Format field is used instead. If the body of the message
  cannot be returned in the return type specified, the message is sent to
  the requeue option if the requeue option is specified; otherwise, the
  read operation fails. MQ does not enforce that when MQMD.Format is
  “MQSTR”, the message body contains only character data.
  Programmers should always specify image or varbinary return types.

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:

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

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

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

In this 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_jms_provider?queue=queue.sample'
MESSAGE PROPERTY 'color=red, red=color')

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

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

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

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

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

select msgsend('Sending message with properties',
'my_jms_provider?queue=queue.sample',
MESSAGE PROPERTY 'color=red, shape=square'
If another client wants to select messages in which the property “color” equals “red” and the property “shape” equals “square”, that client must execute the following:

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

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.

TIBCO JMS only – if message_filter is specified to msgrecv, it is ignored.

MQSeries only – you can select particular messages by specifying the correlation and the message IDs in the message options.

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]

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 4-10 if you are using TIBCO. Use the options in Table 4-11 if you are using MQSeries.
- **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
  ```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>option_string</td>
<td>String describing the option you want to specify.</td>
</tr>
<tr>
<td>simple_identifier</td>
<td>String that identifies the value of an option.</td>
</tr>
<tr>
<td>quoted_string</td>
<td>String formed using the normal SQL conventions for embedded quotation marks.</td>
</tr>
<tr>
<td>integer_literal</td>
<td>Literal specified by normal SQL conventions.</td>
</tr>
<tr>
<td>float_literal</td>
<td>Literal specified by normal SQL conventions.</td>
</tr>
<tr>
<td>Parameter</td>
<td>Description</td>
</tr>
<tr>
<td>---------------</td>
<td>--------------------------------------------------</td>
</tr>
<tr>
<td>true</td>
<td>A Boolean literal.</td>
</tr>
<tr>
<td>false</td>
<td>A Boolean literal.</td>
</tr>
<tr>
<td>null</td>
<td>A null literal.</td>
</tr>
<tr>
<td>byte_literal</td>
<td>Has the form 0xHH, where each H is a hexadecimal digit.</td>
</tr>
</tbody>
</table>
properties_clause

is a property_option_string, or one of the options listed in Table 4-12 on page 100 for MQSeries, and Table 4-13 on page 109 for TIBCO JMS. 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.

TIBCO JMS only – if you use a property not listed in Table 4-13 on page 109, it is set as a property in the message properties of the message sent.

Use the options in Table 4-13 on page 109 for msgsend using TIBCO JMS.

MQSeries only – the values of properties_clause differ based on what you specify in the rhfCommand option:

- The properties in Table 4-14 on page 110 are effective only if rhfCommand is deletePublication.
  
  A deletePublication command message sent to the publication stream instructs the MQ pub/sub broker to delete its copy of any retained publications for the specified topics within the publication stream.
  
  The message_body argument to msgsend is ignored.

- The properties in Table 4-15 on page 111 are effective only if rhfCommand is deregisterPublisher.

- The properties in Table 4-16 on page 112 are effective only if rhfCommand is deregisterSubscriber.
  
  A deregisterPublisher command message sent to the MQ pub/sub broker control queue informs the broker that the publisher will no longer be publishing publications on the topics specified.
  
  The message_body argument to msgsend is ignored.

  If the msgType is request, the reply message is sent to replyToQmgr and replyToQueue.

- The properties in Table 4-17 on page 113 are effective only if rhfCommand is publish.
  
  A publish command message is sent to the publication stream queue to publish information on specific topics. The publication
data is specified as the message_body argument to msgsend.

If the msgType is request, the reply message is sent to replyToQmgr and replyToQueue.

- The properties in Table 4-19 on page 118 are effective only if rhfCommand is registerSubscriber.
  A registerSubscriber command message sent to the MQ pub/sub broker control queue informs the broker that the publisher is publishing, or can, publish data on one or more specified topics. If the publisher is already registered, and there are no other errors, the publisher’s registration is modified accordingly.
  If the msgType is request, the reply message is sent to replyToQmgr and replyToQueue.
- The properties in Table 4-20 on page 121 are effective only if rhfCommand is requestUpdate.
  A requestUpdate command message sent to the MQ pub/sub broker control queue informs the broker that the subscriber wants the broker to forward all retained publications that match the topic specified.
  If the msgType is request, the reply message is sent to replyToQmgr and replyToQueue.

**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.
**msgsend**

**header_clause**
allows users to specify only those header properties that are specified in Table 4-12 on page 100 for MQSeries and Table 4-13 on page 109 for TIBCO JMS. 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** TIBCO JMS – sends the message “Hello” to the specified endpoint:

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

**Example 2** TIBCO JMS – sends the message “Hello Messaging World!” to the specified endpoint:

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

**Example 3** TIBCO JMS – 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:

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

**Example 4** TIBCO JMS – sets two properties and generates an XML schema for the message:

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

**Example 5** TIBCO JMS – shows user-specified values for message properties:

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

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

**Example 6** MQSeries – sends a request message, and the reply is expected on the specified queue, in the same Queue Manager.

```
select msgsend('do something',
    'ibm_mq:chnl1/TCP/host1(5678)?qmgr=QM,queue=QUEUE.COMMAND',
    option 'msgType=request'
MESSAGE PROPERTY 'replyToQueue=QUEUE.REPLY')
```

**Example 7** MQSeries – sends a reply message. The correlation ID, and the reply queue were extracted from a previously received request message:

```
select @correlationId = msgpropvalue("CorrelId", @@msgheader)
select @replyQ = @@msgreplytoinfo
select msgsend('i'm done',
    @replyQ
    option 'msgType=report'
MESSAGE PROPERTY 'correlationId=' + @correlationId)
```

**Example 8** MQSeries – sends a report message. The correlation ID, reply queue, and report message data header were extracted from a previously received request message:

```
select @correlationId = msgpropvalue("CorrelId", @@msgheader)
select @replyQ = @@msgreplytoinfo
select msgsend(@reportData,
    @replyQ
    option 'msgType=report'
MESSAGE PROPERTY 'correlationId=' + @correlationId)
```

**Example 9** MQSeries – sends four datagram messages. Each message is part of the group "theGroup", and each message has an increasing sequence number:

```
begin tran
select msgsend('message 1',
    'ibm_mq:chnl1/TCP/host1(5678)?qmgr=QM,queue=QUEUE.COMMAND',
    MESSAGE PROPERTY 'groupId=theGroup,sequenceId=1')
select msgsend('message 2',
    'ibm_mq:chnl1/TCP/host1(5678)?qmgr=QM,queue=QUEUE.COMMAND',
    MESSAGE PROPERTY 'groupId=theGroup,sequenceId=2')
select msgsend('message 3',
    'ibm_mq:chnl1/TCP/host1(5678)?qmgr=QM,queue=QUEUE.COMMAND',
    MESSAGE PROPERTY 'groupId=theGroup,sequenceId=3')
```
Example 10 MQSeries – sends a datagram message. Various confirmation reports are requested, and they are sent to the “myReplyQueue”:

```sql
select msgsend('message 4',
    'ibm_mq:chnl1/TCP/host1(5678)?qmgr=QM,queue=QUEUE.COMMAND',
    MESSAGE PROPERTY 'groupId=theGroup,sequenceId=4,lastMsgInGroup=yes')
commit
```

Example 11 MQSeries – publishes a datagram message with topics “A”, “A/B”, “A/B/C”. The publisher is registered to publish on topics “A”, “A/B”, and “A/B/C”, and the publication contains information about topic “A/B”. The default MQ pub/sub broker queue and stream queues are used:

```sql
-- First register the publisher
select msgsend(null,
    'ibm_mq:channel1/TCP/host1(5678)?queue=SYSTEM.BROKER.CONTROL.QUEUE
    option 'msgType=datagram,rfhCommand=registerPublisher'
    MESSAGE PROPERTY 'topics=''a:A/B:a/b/c''')

-- Now publish the publication
select msgsend('something about A/B',
    'ibm_mq:channel1/TCP/host1(5678)?queue=SYSTEM.BROKER.DEFAULT.STREAM'
    option 'msgType=datagram,rfhCommand=publish'
    MESSAGE PROPERTY 'topics=A/B'
```

Example 12 MQSeries – sends multiple messages in a group. Since ordering is set to logical, specify only the `msgInGroup`, `lastMsgInGroup`, `msgSegment`, `msgLastSegment` options. The Queue Manager selects a name for the group since it is not specified:

```sql
begin tran
select msgsend('first logical message of the group',
    'ibm_mq:chnl1/TCP/host1(5678)?qmqr=QM,queue=QUEUE.COMMAND',
    MESSAGE PROPERTY 'ordering=logical,msgInGroup=yes')

select msgsend('second logical message of the group',
    'ibm_m_q:chnl1/TCP/host1(5678)?qmqr=QM,queue=QUEUE.COMMAND',
    MESSAGE PROPERTY 'ordering=logical,msgInGroup=yes')
```

Example 10

MQSeries – sends a datagram message. Various confirmation reports are requested, and they are sent to the “myReplyQueue”:

```sql
select msgsend('I want a confirmation',
    'ibm_mq:channel1/TCP/host1(5678)?queue=QUEUE.COMMAND',
    MESSAGE PROPERTY 'replyToQueue=myReplyQueue'
    + ',exceptionReport=yes,'
    + ',arrivalReport=withData'
    + ',deliveryReport=withFullData'
```

Example 11

MQSeries – publishes a datagram message with topics “A”, “A/B”, “A/B/C”. The publisher is registered to publish on topics “A”, “A/B”, and “A/B/C”, and the publication contains information about topic “A/B”. The default MQ pub/sub broker queue and stream queues are used:

```sql
-- First register the publisher
select msgsend(null,
    'ibm_mq:channel1/TCP/host1(5678)?queue=SYSTEM.BROKER.CONTROL.QUEUE
    option 'msgType=datagram,rfhCommand=registerPublisher'
    MESSAGE PROPERTY 'topics=''a:A/B:a/b/c''')

-- Now publish the publication
select msgsend('something about A/B',
    'ibm_mq:channel1/TCP/host1(5678)?queue=SYSTEM.BROKER.DEFAULT.STREAM'
    option 'msgType=datagram,rfhCommand=publish'
    MESSAGE PROPERTY 'topics=A/B'
```
Example 13 Uses `msgsend` to register, then deregister a subscriber. The subscriber is interested in all publications that match the topics “A” or “A/B/**”. Matching publications are forwarded to the queue “Q2” by the MQ pub/sub broker:

```
-- Register the subscriber
select msgsend(null,
  'ibm_mq:channel1/TCP/host1(5678)' + '?qmgr=QM,queue=SYSTEM.BROKER.CONTROL.QUEUE'
  option 'msgType=datagram,rfhCommand=registerSubscriber'
  MESSAGE PROPERTY 'topics=''A:A/B/**'',streamName=stream1,queueName=Q2')

-- Publish a message to the stream queue, let it do implicit registration
select msgsend('happy birthday',
  'ibm_mq:channel1/TCP/host1(5678)?qmgr=QM,
  queue=stream1'
  option 'msgType=datagram,rfhCommand=publish'
  MESSAGE PROPERTY 'topics='''A'''')

-- Read a message forwarded to us by the MQ pub/sub
select msgrecv(
  'ibm_mq:channel1/TCP/host1(5678)?qmgr=QM,
  queue=Q2'
  option 'timeout=50ss')

-- Deregister the subscriber
select msgsend(null,
  'ibm_mq:channel1/TCP/host1(5678)' + '?qmgr=QM,queue=SYSTEM.BROKER.CONTROL.QUEUE'
  option 'msgType=datagram,rfhCommand=deregisterSubscriber'
```
**msgsend**

**MESSAGE PROPERTY**  `topics=''A:A/B/*'', streamName=stream1, queueName=Q2'``

**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 4-3 on page 40.

- 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_jms_provider?queue=queue.sample',
                MESSAGE PROPERTY 'priority=''high'', priority=yes, priority=7')
```

- After you execute `msgsend`, the values of the global variables are set with information for that call. For more details, see “Message-related global variables” on page 32.
• 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.

**Note** This behavior is new with version 12.5.3a, and differs from previous versions.

`msgsend` option `option_string` parameter values

Table 4-10 lists the available `msgsend` option parameters for TIBCO.

<table>
<thead>
<tr>
<th>Types</th>
<th>Values</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>schema</td>
<td>no</td>
<td>no</td>
<td>• <code>user_schema</code> is a user-supplied schema describing the message_body.</td>
</tr>
<tr>
<td></td>
<td>yes</td>
<td></td>
<td>• no indicates that no schema is generated and sent out as part of the message.</td>
</tr>
<tr>
<td></td>
<td>&quot;user_schema&quot;</td>
<td></td>
<td>• yes indicates that Adaptive Server generates an XML schema for the message. yes is meaningful only in a message_body that uses the parameter <code>select_for_xml</code>. <code>select_for_xml</code> 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.</td>
</tr>
<tr>
<td>type</td>
<td>text, bytes</td>
<td>text</td>
<td>The type of message to send.</td>
</tr>
</tbody>
</table>

Table 4-11 lists the available `msgsend` option parameters for MQSeries.
### Table 4-11: Valid MQSeries option string types and values for msgsend

<table>
<thead>
<tr>
<th>Types</th>
<th>Values</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
</table>
| msgType | • datagram  
         | • request  
         | • reply  
         | • report | datagram | If the type of the message is:  
• request – you must also specify the replyQueue property.  
• report – you must also specify the reportDataHeader and feedback properties. |
MQRF headers, for MQ pub/sub, are control messages that are sent to a queue and read by the MQ pub/sub broker. The broker acts upon the message it reads from the queue.

If rfhCommand is null, the message does not include the MQRF header. The message includes the MQRF header with any other value for rfhCommand, with the MQPSCommand set to the following:

- deletePublication – set to DeletePub. The endpoint is the endpoint to the DeletePub stream queue. See Table 4-14 on page 110.
- deregisterPublisher – set to DeregPub. See Table 4-15 on page 111.
- deregisterSubscriber – set to DeleteSub. See Table 4-16 on page 112.
- publish – set to Publish. The endpoint is the endpoint to the publishing stream queue. See Table 4-17 on page 113.
- registerPublisher – set to RegPub. See “msgsend properties if rfhCommand is set to deletePublications” on page 110.
- registerSubscriber – set to RegSub. See Table 4-19 on page 118.
- requestUpdate – set to ReqUpdate. See Table 4-20 on page 121.

The message is sent to the endpoint you specify. For these options, specify the endpoint to the publishing stream queue:

- publish
- deletePublication

For these options, specify the endpoint to the MQ pub/sub broker control queue:

- deregisterPublisher
- deregisterSubscriber
- registerPublisher
- registerSubscriber
- requestUpdate

<table>
<thead>
<tr>
<th>Types</th>
<th>Values</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>rfhCommand</td>
<td>• null</td>
<td>null</td>
<td>MQRF headers, for MQ pub/sub, are control messages that are sent to a queue and read by the MQ pub/sub broker. The broker acts upon the message it reads from the queue. If rfhCommand is null, the message does not include the MQRF header. The message includes the MQRF header with any other value for rfhCommand, with the MQPSCommand set to the following:</td>
</tr>
<tr>
<td></td>
<td>• deletePublication</td>
<td></td>
<td>• deletePublication – set to DeletePub. The endpoint is the endpoint to the DeletePub stream queue. See Table 4-14 on page 110.</td>
</tr>
<tr>
<td></td>
<td>• deregisterPublisher</td>
<td></td>
<td>• deregisterPublisher – set to DeregPub. See Table 4-15 on page 111.</td>
</tr>
<tr>
<td></td>
<td>• deregisterSubscriber</td>
<td></td>
<td>• deregisterSubscriber – set to DeleteSub. See Table 4-16 on page 112.</td>
</tr>
<tr>
<td></td>
<td>• publish</td>
<td></td>
<td>• publish – set to Publish. The endpoint is the endpoint to the publishing stream queue. See Table 4-17 on page 113.</td>
</tr>
<tr>
<td></td>
<td>• registerPublisher</td>
<td></td>
<td>• registerPublisher – set to RegPub. See “msgsend properties if rfhCommand is set to deletePublications” on page 110.</td>
</tr>
<tr>
<td></td>
<td>• registerSubscriber</td>
<td></td>
<td>• registerSubscriber – set to RegSub. See Table 4-19 on page 118.</td>
</tr>
<tr>
<td></td>
<td>• requestUpdate</td>
<td></td>
<td>• requestUpdate – set to ReqUpdate. See Table 4-20 on page 121.</td>
</tr>
</tbody>
</table>
msgsend properties_clause parameter values

Table 4-12 lists the available msgsend properties_clause parameters for MQSeries.

<table>
<thead>
<tr>
<th>Types</th>
<th>Values</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>arrivalReport</td>
<td>• yes</td>
<td>no</td>
<td>Arrival of this message to the final destination should generate a confirm-on-arrival (COA) report. You must specify replyToQueue. If you specify:</td>
</tr>
<tr>
<td></td>
<td>• withData</td>
<td></td>
<td>• yes — the COA report generates without data from the received message.</td>
</tr>
<tr>
<td></td>
<td>• withFullData</td>
<td></td>
<td>• withData — the COA report generates with the first 100 bytes of the data from the received message.</td>
</tr>
<tr>
<td></td>
<td>• no</td>
<td></td>
<td>• withFullData — the COA report generates with the full data from the received message.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• no — the COA report is not generated.</td>
</tr>
<tr>
<td>correlationId</td>
<td>• null</td>
<td>null</td>
<td>Clients set correlation ID to link messages together. MQ limits this string to 24 bytes.</td>
</tr>
<tr>
<td></td>
<td>• string</td>
<td></td>
<td>MQ defines this field as unsigned char, which indicates that it can support binary values. To enter a binary string as the correlationId, use “0x…” as the value.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Do not use quotes around the value.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>If rfhCommand is not null:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• If correlationId is not null, a new correlation ID is not requested. If correlationAsId is yes, and correlationId is null, this is a separate traditional identity (one where correlation ID is empty).</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• For rfhCommands of deletePublication, deregisterPublisher, publish, and registerPublisher, the correlation ID specified is as part of the publisher’s traditional identity.</td>
</tr>
<tr>
<td>Types</td>
<td>Values</td>
<td>Default</td>
<td>Description</td>
</tr>
<tr>
<td>---------------</td>
<td>-------------------------</td>
<td>---------</td>
<td>--------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>deliveryReport</td>
<td>• yes</td>
<td>no</td>
<td>Delivery of this message from the final destination generates a confirm-on-delivery (COD) report. You must specify replyToQueue. If:</td>
</tr>
<tr>
<td></td>
<td>• withData</td>
<td></td>
<td>• yes – the COA report generates without data from the received message.</td>
</tr>
<tr>
<td></td>
<td>• withFullData</td>
<td></td>
<td>• withData – the COA report generates with the first 100 bytes of the data from the received message.</td>
</tr>
<tr>
<td></td>
<td>• no</td>
<td></td>
<td>• withFullData – the COA report generates with the full data from the received message.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• no – the COA report is not generated.</td>
</tr>
<tr>
<td>exceptionReport</td>
<td>• yes</td>
<td>no</td>
<td>Expiration of this message or failure of this send generates an exception report. You must specify replyToQueue. If:</td>
</tr>
<tr>
<td></td>
<td>• withData</td>
<td></td>
<td>• yes – the exception report generates without data from the received message.</td>
</tr>
<tr>
<td></td>
<td>• withFullData</td>
<td></td>
<td>• withData – the exception report generates with the first 100 bytes of the data from the received message.</td>
</tr>
<tr>
<td></td>
<td>• no</td>
<td></td>
<td>• withFullData – the exception report generates with the full data from the received message.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• no – the exception report is not generated.</td>
</tr>
<tr>
<td>expirationReport</td>
<td>• yes</td>
<td>no</td>
<td>The failure of this send generates an exception report. You must specify replyToQueue. If:</td>
</tr>
<tr>
<td></td>
<td>• withData</td>
<td></td>
<td>• yes – the exception report generates without data from the received message.</td>
</tr>
<tr>
<td></td>
<td>• withFullData</td>
<td></td>
<td>• withData – the exception report generates with the first 100 bytes of the data from the received message.</td>
</tr>
<tr>
<td></td>
<td>• no</td>
<td></td>
<td>• withFullData – the exception report generates with the full data from the received message.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• no – the exception report is not generated.</td>
</tr>
</tbody>
</table>
### Types: `expiry`
- **Values**: timespec between -1, 0-\((2^{32} - 1)\)
- **Default**: -1, no expiration
- **Description**: The message’s time-to-live on the Queue Manager. Units are in milliseconds if the timespec is an integer.
Values are:
- 0 – message does not expire.
- -1 – uses the default defined for the queue.

**Note** The MQ expiry is in tenths of a second, so this number is rounded to the tenths of a second before being passed to MQ.

See timespec on page 135 for more information.

### Types: `feedback`
- **Values**: integer
  - **Default**: 0
  - **Description**: For report messages, feedback is a code that indicates the nature of the report message.
  - MQ defines one feedback code range each for:
    - System report messages
      - MQFB_APPL_FIRST (65536) to MQFB_APPL_LAST (999999999)
    - Application report messages

### Types: `formatName`
- **Values**: null, string
- **Default**: null
- **Description**: Application-defined property to pass information about the message formats.
  - This property allows sending applications to set a format name that describes the message data.
  - A receiving application can check `formatName` in `@@msgheader` to decide how to process the message data.
  - Names beginning with “MQ” are reserved.
  - MQ limits this string to 8 bytes.

### Types: `groupId`
- **Values**: null, string
- **Default**: null
- **Description**: User-defined group.
  - MQ limits this string to 24 bytes.
  - MQ defines this field as unsigned char, which indicates that it can support binary values. To enter a binary string as the `groupId`, use “0x…” as the value.
  - Do not use quotes around the value, or it is interpreted as a quoted string.
  - If `groupId` is not specified and one of the grouping properties is specified, the Queue Manager generates the group name.
  - Ignored if `ordering` is set to logical.
  - All messages of a group must be sent in the same transaction.
### CHAPTER 4  SQL Reference

---

#### lastMsgInGroup

- **Types:** string
- **Values:** `yes`, `no`
- **Default:** `no`
- **Description:**
  - If the value is `yes`, marks a message as being the last logical message of a group.
  - To have a single logical message in a group by itself, you must set `lastMsgInGroup` to `yes`.
  - You must send all messages of a group in the same transaction.

#### mode

- **Types:** string
- **Values:** `persistent`, `non-persistent`, `default`
- **Default:** `default`
- **Description:**
  - If mode is:
    - `persistent` – the message is backed by the messaging 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.
    - `default` – the default defined for the queue is used.

#### msgId

- **Types:** string
- **Values:** `null`, `string`
- **Default:** `null`
- **Description:**
  - When specified, WebSphere MQSeries replaces any existing message ID with the value specified for `msgId`.
  - MQ limits this string to 24 bytes.
  - MQ defines this field as “unsigned char,” which indicates that it can support binary values.
  - To enter a binary string as the `msgId`, use “0x...” as the value. Do not use quotes around the value.

#### msgInGroup

- **Types:** string
- **Values:** `yes`, `no`
- **Default:** `no`
- **Description:**
  - If the value is `yes`, this message is a logical message of a message group.
  - For messages in a group, you must set this property to `yes` for all logical messages of the group, except the last one, which should have `lastMsgInGroup` set to `yes`.
  - You must send all messages of a group in the same transaction.

#### msgLastSegment

- **Types:** string
- **Values:** `yes`, `no`
- **Default:** `no`
- **Description:**
  - If the value is `yes`, this message is the last segment of a segmented message. To have a segment message in a local message by itself, the message must have `msgLastSegment` set to `yes`.
  - When the value is `yes` and ordering is set to `physical`, you must also set the `offset` property.
  - You must send all messages of a group in the same transaction.
If the value is yes, this message is a segment of a segmented message. For messages that are part of a single segment, you must set this property to yes for all segments except the last one, which should be have msgLastSegment set to yes.

When the value is yes and ordering is set to physical, you must also set the offset property.

You must send all messages in a group in the same transaction.

If you specify replyToQueue, If:

- yes – when the retrieving application reads this message and acts negatively on it, a negative-action (NAN) report is generated.
- no – the NAN report is not generated.

When the message is a segment of a segmented message, you should set offset to the byte offset of the current message within the logical message.

-1 indicates that the offset is not specified.

offset is ignored unless msgSegment, or msgLastSegment are also specified.

Ignored by msgpublish.

Ignored if ordering is set to logical.

You must send all messages of a group in the same transaction.

When this property is:

- physical – the application can send messages that are part of a group (or segmented message) in any order. The Queue Manager returns errors if it detects missing segments, or holes in the sequence Identifiers.
- logical – the application needs only to set the msgInGroup, lastMsgInGroup, msgSegment, and lastMsgSegment options appropriately. The Queue Manager automatically sets the group name, sequence identifier, and segment offset.

<table>
<thead>
<tr>
<th>Types</th>
<th>Values</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>msgSegment</td>
<td>• yes</td>
<td>no</td>
<td>If the value is yes, this message is a segment of a segmented message. For</td>
</tr>
<tr>
<td></td>
<td>• no</td>
<td></td>
<td>messages that are part of a single segment, you must set this property to</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>yes for all segments except the last one, which should be have msgLastSegment</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>set to yes.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>When the value is yes and ordering is set to physical, you must also set</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>the offset property.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>You must send all messages in a group in the same transaction.</td>
</tr>
<tr>
<td>negativeActionReport</td>
<td>• yes</td>
<td>no</td>
<td>You must specify replyToQueue. If:</td>
</tr>
<tr>
<td></td>
<td>• no</td>
<td></td>
<td>• yes – when the retrieving application reads this message and acts</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>negatively on it, a negative-action (NAN) report is generated.</td>
</tr>
<tr>
<td>offset</td>
<td>integer</td>
<td>-1</td>
<td>When the message is a segment of a segmented message, you should set offset</td>
</tr>
<tr>
<td></td>
<td>between -1, 0</td>
<td></td>
<td>to the byte offset of the current message within the logical message.</td>
</tr>
<tr>
<td></td>
<td>maxint</td>
<td></td>
<td>-1 indicates that the offset is not specified.</td>
</tr>
<tr>
<td>offset</td>
<td></td>
<td></td>
<td>offset is ignored unless msgSegment, or msgLastSegment are also specified.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Ignored by msgpublish.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Ignored if ordering is set to logical.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>You must send all messages of a group in the same transaction.</td>
</tr>
<tr>
<td>onNoDelivery</td>
<td>• deadLetter</td>
<td>deadLetter</td>
<td>If:</td>
</tr>
<tr>
<td></td>
<td>• discard</td>
<td></td>
<td>• deadLetter – if the message cannot be delivered, the message is put on</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>the dead-letter queue.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• discard – the message is discarded by the Queue Manager.</td>
</tr>
<tr>
<td>ordering</td>
<td>• logical</td>
<td>physical</td>
<td>When this property is:</td>
</tr>
<tr>
<td></td>
<td>• physical</td>
<td></td>
<td>• physical – the application can send messages that are part of a group</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>(or segmented message) in any order. The Queue Manager returns errors if it</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>detects missing segments, or holes in the sequence Identifiers.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• logical – the application needs only to set the msgInGroup, lastMsgInGroup,</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>msgSegment, and lastMsgSegment options appropriately. The Queue Manager</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>automatically sets the group name, sequence identifier, and segment offset.</td>
</tr>
</tbody>
</table>

104 Adaptive Server Enterprise
### Types

<table>
<thead>
<tr>
<th>Types</th>
<th>Values</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
</table>
| positiveActionReport | • yes  
                    | • no | no            | You must specify replyToQueue. If:                                                                                                          |
| priority           | integer:                    | -1      | Controls the priority of the message. If:                                                                                                |
|                   | • -1.                        |         | -1 – the default priority as defined for the queue is used.                                                                             |
|                   | • 0 to queue manager        |         | priority specified is greater than the max priority defined for the Queue Manager – the max priority defined for the Queue Manager is used. This is implemented by MQ. |
| replyCorrelationId | • msgId                      | msgId   | If:                                                                                                                                     |
|                   | • correlationId              |         | • msgId – the correlation ID in the report message should use the message ID of the received message.                                   |
|                   |                             |         | • correlationId – the correlation ID in the report message uses the correlation ID of the received message.                             |
| replyMsgId         | • new                        | new     | If:                                                                                                                                     |
|                   | • original                   |         | • new – the generated report message contains a new message ID.                                                                        |
|                   |                             |         | • original – the report message should use the same message ID as the message received.                                                 |
**msgsend**

<table>
<thead>
<tr>
<th>Types</th>
<th>Values</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
</table>
| replyToInputMode    | • browse                      | Qdefault| The mode that the replyToQueue is opening. When you specify replyToQueue, the queue is automatically opened for subsequent input. This mode specifies the input mode that the replyToQueue is opening. This property is ignored if you do not specify replyToQueue. The modes have the following meanings:  
  • browse – the queue is opened for browsing only. You get an error from the Queue Manager if you attempt to do a destructive read.  
  • Qdefault – the queue is opened in the default input mode as defined for the queue.  
  • shared – the queue is opened in shared input mode. You see an error if the queue is already opened in exclusive mode by another MQ handle.  
  • exclusive – the queue is opened in exclusive input mode. You see an error if the queue is already opened in shared or exclusive mode by another MQ handle.  
  • browse+Qdefault – the queue is opened for browsing, as well as for the default input mode as defined for the queue.  
  • browse+shared – the queue is opened for browsing, as well as for shared input mode. You see an error if the queue is already opened in exclusive mode by another MQ handle.  
  • browse+exclusive – the queue is opened for browsing, as well as for exclusive input mode. You see an error if the queue is already opened in shared or exclusive mode by another MQ handle. |
|                     | • Qdefault                    |         |                                                                köö                                                                                                                      |
|                     | • shared                      |         |                                                                köö                                                                                                                      |
|                     | • exclusive                   |         |                                                                köö                                                                                                                      |
|                     | • browse+Qdefault             |         |                                                                köö                                                                                                                      |
|                     | • browse+shared               |         |                                                                köö                                                                                                                      |
|                     | • browse+exclusive            |         |                                                                köö                                                                                                                      |
| replyToModel        | • null                        | null    | The name of the model queue from which the reply queue is created, when the replyToQueue is a dynamic queue. If you do not specify replyToQueue, this property is ignored. MQ limits this string to 48 bytes. |
|                     | • string                      |         |                                                                köö                                                                                                                      |
**replyToQmgr**

- null
- string

The Queue Manager where `replyToQueue` resides.

If you do not specify `replyToQueue`, this property is ignored.

MQ limits this string to 48 bytes.

**replyToQueue**

- null
- string

The queue where the application expects a reply to a request message.

**Note** The message type sent does not have to be request, as MQ does not enforce this.

If `replyToQmgr` is not specified, `replyToQueue` is assumed to be the same Queue Manager as the current endpoint.

If the queue name specified ends with a `*`, a system-generated dynamic queue name is generated with the specified prefix.

If `replyToModel` and a dynamic queue name are specified, the dynamic queue is created from the model queue specified for `replyToModel`.

You can obtain system-generated dynamic queue names after the send operation via the `@@msgreplytoinfo` session variable.

**Note** When you specify a dynamic queue name, the current Adaptive Server login must have “crt” authorization in the Queue Manager to create the dynamic queue.

When a dynamic queue name is specified, you must manually delete the dynamic queue that is created if the receiving application does not do so.

When `rfhCommand` is not null, you can specify `replyToQueue` to get responses from the MQ pub/sub broker.
MQRF headers, for MQ pub/sub, are control messages that are sent to a queue and read by the MQ pub/sub broker. The broker acts upon the message that it reads from the queue.

If `rfhCommand` is null, the message does not include the MQRF header. The message includes the MQRF header with any other value for `rfhCommand`, with the MQPSCommand set to the following:

- `deletePublication` – set to `DeletePub`. The endpoint is the endpoint to the publishing stream queue. See Table 4-14 on page 110.
- `deregisterPublisher` – set to `DeregPub`. See Table 4-15 on page 111.
- `deregisterSubscriber` – set to `DeleteSub`. See Table 4-16 on page 112.
- `publish` – set to `Publish`. The endpoint is the endpoint to the publishing stream queue. See Table 4-17 on page 113.
- `registerPublisher` – set to `RegPub`. See “msgsend properties if `rfhCommand` is set to `deletePublications`” on page 110.
- `registerSubscriber` – set to `RegSub`. See “msgsend properties if `rfhCommand` is set to `deletePublications`” on page 110.
- `requestUpdate` – set to `ReqUpdate`. See “msgsend properties if `rfhCommand` is set to `deletePublications`” on page 110.

The message is sent to the endpoint you specify. For these options, specify the endpoint to the publishing stream queue:

- `publish`
- `deletePublication`

For these options, specify the endpoint to the MQ pub/sub broker control queue:

- `deregisterPublisher`
- `deregisterSubscriber`
- `registerPublisher`
- `registerSubscriber`
- `requestUpdate`
Table 4-13 lists the available `msgsend properties_clause` parameters for TIBCO JMS.

Table 4-13: Valid TIBCO JMS message property `properties_option_string` types and values for `msgsend`

<table>
<thead>
<tr>
<th>Option</th>
<th>Values</th>
<th>Default</th>
<th>Disposition</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>sequenceId</code></td>
<td><code>integer</code> between -1 – 9,999,999</td>
<td>-1</td>
<td>header</td>
<td>Used to sequence logical messages that are part of a group. -1 indicates that the sequenceId is not specified. <code>sequenceId</code> is ignored unless <code>msgInGroup</code> or <code>lastMsgInGroup</code> are also specified. Ignored by <code>msgpublish</code>. Ignored if <code>ordering</code> is set to logical. You must send all messages of a group in the same transaction.</td>
</tr>
<tr>
<td><code>ttl</code></td>
<td>0 - (2^{31} - 1)</td>
<td>0</td>
<td>header</td>
<td>(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. <code>ttl</code> uses the <code>timespec</code> option. See <code>timespec</code> on page 135 for more information on <code>timespec</code>.</td>
</tr>
<tr>
<td><code>priority</code></td>
<td>1 to 10</td>
<td>4</td>
<td>header</td>
<td>The behavior of <code>priority</code> 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.</td>
</tr>
<tr>
<td><code>correlation</code></td>
<td><code>string</code></td>
<td>none</td>
<td>header</td>
<td>Client applications set correlation IDs to link messages together. Adaptive Server sets the correlation ID the application specifies.</td>
</tr>
</tbody>
</table>
### msgsend Properties and rfhCommand

For MQSeries, properties in Table 4-14 are effective only if rfhCommand is set to `deletePublication`.

**Table 4-14: msgsend Properties if rfhCommand is set to deletePublications**

<table>
<thead>
<tr>
<th>Property</th>
<th>Values</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>local</td>
<td>• yes</td>
<td>no</td>
<td>If:</td>
</tr>
<tr>
<td></td>
<td>• no</td>
<td></td>
<td>• yes – only the retained publications published locally at this broker are deleted.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• no – globally retained publications are deleted from all brokers in the network.</td>
</tr>
<tr>
<td>streamName</td>
<td>• null</td>
<td>null</td>
<td>Name of the publication stream for the specified topics.</td>
</tr>
<tr>
<td></td>
<td>• string</td>
<td></td>
<td>If not specified, the default is the stream queue to which this MQRFH command message is sent.</td>
</tr>
</tbody>
</table>

MQ limits this string to 48 bytes.
For MQSeries, properties in Table 4-14 are effective only if `rhfCommand` is `deregisterPublisher`.

### Table 4-15: msgsend properties if `rhfCommand` is set to `deregisterPublisher`

<table>
<thead>
<tr>
<th>Property</th>
<th>Values</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>topics</td>
<td>string</td>
<td>none</td>
<td>Use the format detailed in “Syntax for topics” on page 16. Retained messages matching this topic are deleted. At least one topic must be supplied. This is a required property, and is an error if omitted.</td>
</tr>
<tr>
<td>deregAll</td>
<td>• yes</td>
<td>no</td>
<td>If: • yes – all topics registered for this publisher are deregistered, and the topics property is ignored. • no – no registered topics are deregistered. Adaptive Server returns an error if you specify topics.</td>
</tr>
<tr>
<td>streamName</td>
<td>• null</td>
<td>null</td>
<td>If: • Not null – this is the name of the publication stream. • null – <code>SYSTEM.BROKER.DEFAULT.STREAM</code> is assumed. MQ limits this string to 48 bytes.</td>
</tr>
<tr>
<td>topics</td>
<td>• null</td>
<td>null</td>
<td>Use the format detailed in “Syntax for topics” on page 16. These are the topics that this publisher deregisters. Adaptive Server returns an error if: • The <code>deregAll</code> property is set to yes. • topics is not null.</td>
</tr>
<tr>
<td>qmgrName</td>
<td>• null</td>
<td>null</td>
<td>This is the publisher’s Queue Manager name, used to establish the publisher’s traditional identity. Specify it as the same value you specified when you registered the publisher. If null, defaults to <code>replyToQmgr</code>.</td>
</tr>
</tbody>
</table>
For MQSeries, the properties in Table 4-16 are effective only if `rhfCommand` is `deregisterSubscriber`.

**Table 4-16: `msgsend` properties if `rhfCommand` is set to `deregisterSubscriber`**

<table>
<thead>
<tr>
<th>Property</th>
<th>Values</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>queueName</td>
<td>• null</td>
<td>null</td>
<td>This is the publisher’s queue name, used to establish the traditional identity of the publisher. Specify it as the same value you specified when you registered the publisher. If null, defaults to the replyToQueue.</td>
</tr>
<tr>
<td></td>
<td>• string</td>
<td></td>
<td></td>
</tr>
<tr>
<td>correlationAsId</td>
<td>• yes</td>
<td>no</td>
<td>If:</td>
</tr>
<tr>
<td></td>
<td>• no</td>
<td></td>
<td>• yes – correlationId is used as part of the publisher’s traditional identity. You must specify correlationId, but not as 0x00.</td>
</tr>
<tr>
<td></td>
<td>• generate</td>
<td></td>
<td>• no – correlationId is not used as part of the publisher’s traditional identity.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• generate – a system-generated correlationId is used as part of the publisher’s traditional identity.</td>
</tr>
<tr>
<td>deregAll</td>
<td>• yes</td>
<td>no</td>
<td>If:</td>
</tr>
<tr>
<td></td>
<td>• no</td>
<td></td>
<td>• yes – all topics for this subscriber are deregistered. The topics property is ignored.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• no – no subscriber topics are deregistered. Adaptive Server returns an error if topics are not null</td>
</tr>
<tr>
<td>streamName</td>
<td>• null</td>
<td>null</td>
<td>If:</td>
</tr>
<tr>
<td></td>
<td>• string</td>
<td></td>
<td>• Not null – this is the name of the publication stream.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• null – <code>SYSTEM.BROKER.DEFAULT.STREAM</code> is assumed.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>MQ limits this string to 48 bytes.</td>
</tr>
</tbody>
</table>
For MQSeries, the properties in Table 4-17 are effective only if `rhfCommand` is publish.

**Table 4-17: msgsend properties if rhfCommand is set to publish**

<table>
<thead>
<tr>
<th>Property</th>
<th>Values</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>topics</td>
<td><code>string</code></td>
<td>null</td>
<td>Use the format detailed in “Syntax for topics” on page 16. These are the topics on which this publication has information. Wildcards are not allowed. These are the topics on which this publication has information. This is a required property, and an error if omitted.</td>
</tr>
<tr>
<td>qmgrName</td>
<td><code>string</code></td>
<td>null</td>
<td>This is the subscriber’s Queue Manager name, used to establish the traditional identity of the subscriber. You should specify it as the same value that was specified when you registered the subscriber. If null, it defaults to the <code>replyToQmgr</code>.</td>
</tr>
<tr>
<td>queueName</td>
<td><code>string</code></td>
<td>null</td>
<td>This is the subscriber’s queue name, used to establish the traditional identity of the subscriber. You should specify it as the same value that was specified when you registered the subscriber. If null, it defaults to the <code>replyToQueue</code>.</td>
</tr>
<tr>
<td>correlationAsId</td>
<td><code>yes</code>, <code>no</code>, <code>generate</code></td>
<td>no</td>
<td>If:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• <code>yes</code> – <code>correlationId</code> is used as part of the publisher’s traditional identity. You must specify <code>correlationId</code>, but not as 0x00.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• <code>no</code> – <code>correlationId</code> is not used as part of the publisher’s traditional identity.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• <code>generate</code> – a system-generated <code>correlationId</code> is used as part of the publisher’s traditional identity.</td>
</tr>
</tbody>
</table>

Adaptive Server returns an error if:

- `deregAll` is Yes.
- `topics` are not null.

These are the topics that this subscriber deregisters.

If null, it defaults to the `replyToQmgr`.
msgsend

<table>
<thead>
<tr>
<th>Property</th>
<th>Values</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>anom</strong></td>
<td>* yes</td>
<td>no</td>
<td>If:</td>
</tr>
<tr>
<td></td>
<td>* no</td>
<td></td>
<td>* yes – the identity of the publisher is not divulged by the MQ pub/sub broker. Ignored if noReg is yes.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>* no – the identity of the publisher is divulged by the MQ pub/sub broker.</td>
</tr>
<tr>
<td><strong>local</strong></td>
<td>* yes</td>
<td>no</td>
<td>If:</td>
</tr>
<tr>
<td></td>
<td>* no</td>
<td></td>
<td>* yes – the MQ pub/sub broker sends this publication only to subscribers that registered specifying local. Ignored if noReg is yes.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>* no – the MQ pub/sub broker sends this publication to all subscribers.</td>
</tr>
<tr>
<td><strong>directReq</strong></td>
<td>* yes</td>
<td>no</td>
<td>If:</td>
</tr>
<tr>
<td></td>
<td>* no</td>
<td></td>
<td>* yes – the publisher is willing to accept direct request for publication information from other applications. Ignored if noReg is yes. Do not set this option to yes if the anom property is also set to yes, since the MQ pub/sub broker responds with an error.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>* no – the publisher is not willing to accept direct request for publication information from other applications.</td>
</tr>
<tr>
<td><strong>noReg</strong></td>
<td>* yes</td>
<td>no</td>
<td>If the publisher is not already registered with the MQ pub/sub broker as a publisher for this stream and topic and the value of NoReg is:</td>
</tr>
<tr>
<td></td>
<td>* no</td>
<td></td>
<td>* yes – the MQ pub/sub broker does not perform an implicit registration. The anom, local, and directReq properties are ignored.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>* no – the MQ pub/sub broker performs an implicit registration, using the values set by anom, local, and directReq.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>If the publisher is already registered, and anom, local, or directReq are set to yes, the existing registration is altered according to those properties.</td>
</tr>
<tr>
<td>Property</td>
<td>Values</td>
<td>Default</td>
<td>Description</td>
</tr>
<tr>
<td>-------------</td>
<td>-------------------------</td>
<td>---------</td>
<td>--------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>otherSubsOnly</td>
<td>yes, no</td>
<td>no</td>
<td>If:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• yes – the MQ pub/sub sends this publication to this publisher if this publisher has a subscription on this publication.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• no – the MQ pub/sub broker does not send this publication to this publisher, even if this publisher has a subscription on this publication.</td>
</tr>
<tr>
<td>publishSequenceId</td>
<td>number between -1, 0–(2^32 – 1)</td>
<td>-1</td>
<td>If:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Not -1, this is the sequence number of the publication. It should increase with each publication, but the MQ pub/sub broker does not validate it.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• If -1, the sequence number is not set.</td>
</tr>
<tr>
<td>publishTimeStamp</td>
<td>null, integer</td>
<td>null</td>
<td>If:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Not null, this is the publication timestamp in the form of YYYYMMDDHHMMSSSTH, using universal time. The format is not validated.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• null – the publication timestamp is not set.</td>
</tr>
<tr>
<td>qmgrName</td>
<td>null, string</td>
<td>null</td>
<td>This is the Queue Manager used to determine the publisher’s traditional identity. This is also where subscribers can send direct requests to this publisher. MQ limits this string to 48 bytes.</td>
</tr>
<tr>
<td>queueName</td>
<td>null, string</td>
<td>null</td>
<td>This is the queue used to determine the publisher’s traditional identity. This is also where subscribers can send direct requests to this publisher. MQ limits this string to 48 bytes.</td>
</tr>
<tr>
<td>retainPub</td>
<td>yes, no</td>
<td>no</td>
<td>If:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• yes – the MQ pub/sub broker does not send this publication to this publisher, even if this publisher has a subscription on this publication.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• no – the MQ pub/sub sends this publication to this publisher if this publisher has a subscription on this publication.</td>
</tr>
</tbody>
</table>
### msgsend

**Property** | **Values** | **Default** | **Description**
--- | --- | --- | ---
stringData | • null  
• string | null | If not null, this is optional publisher-defined information that is included in the publication’s MQRF header.  

**Note** Although MQ pub/sub allows multiple stringData tags in the MQRF header, RTMS supports only one.

integerData | *number* between -1, 0–(2**32** – 1) | -1 | If not -1, this is optional publisher-defined information that is included in the publication’s MQRF header.  

**Note** Although MQ pub/sub allows multiple integerData tags in the MQRF header, RTMS supports only one.

correlationAsId | • yes  
• no  
• generate | no | If:

• yes – correlationId is used as part of the publisher’s traditional identity. You must specify correlationId, but not as 0x00.  

• no – correlationId is not used as part of the publisher’s traditional identity.  

• generate – a system-generated correlationId is used as part of the publisher’s traditional identity.

For MQSeries the properties in Table 4-18 are effective only if rhfCommand is registerPublisher.

### Table 4-18: MQSeries msgsend properties if rhfCommand is set to registerPublisher

<table>
<thead>
<tr>
<th>Property</th>
<th>Values</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
</table>
anon | • yes  
• no | no | If:

• yes – MQ pub/sub broker does not divulge the identity of the publisher.  

• no – MQ pub/sub broker divulges the identity of the publisher.
### Property Values Default Description

<table>
<thead>
<tr>
<th>Property</th>
<th>Values</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>correlationAsId</td>
<td>yes, no, generate</td>
<td>no</td>
<td>If:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• yes – correlationId is used as part of the publisher’s traditional identity. You must specify correlationId, but not as 0x00.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• no – correlationId is not used as part of the publisher’s traditional identity.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• generate – a system-generated correlationId is used as part of the publisher’s traditional identity.</td>
</tr>
<tr>
<td>directReq</td>
<td>yes, no</td>
<td>no</td>
<td>If:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• yes – the publisher is willing to accept direct request for publication information from other applications.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• no – the publisher is not willing to accept direct request for publication information from other applications.</td>
</tr>
<tr>
<td>local</td>
<td>yes, no</td>
<td>no</td>
<td>If:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• yes – the MQ pub/sub broker only sends this publication to subscribers that registered specifying Local.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• no – the MQ pub/sub broker sends this publication to all subscribers.</td>
</tr>
<tr>
<td>qmgrName</td>
<td>null, string</td>
<td>null</td>
<td>This is the Queue Manager used to determine the publisher’s traditional identity. This is also where subscribers can send Direct Request requests to this publisher.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>MQ limits this string to 48 bytes.</td>
</tr>
<tr>
<td>queueName</td>
<td>null, string</td>
<td>null</td>
<td>This is the queue used to determine the publisher’s traditional identity. This is also where subscribers can send Direct Request requests to this publisher.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>MQ limits this string to 48 bytes.</td>
</tr>
</tbody>
</table>
For MQSeries the properties in Table 4-19 are effective only if `rhfCommand` is `registerSubscriber`.

<table>
<thead>
<tr>
<th>Property</th>
<th>Values</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>streamName</td>
<td><code>null</code></td>
<td>null</td>
<td>If:</td>
</tr>
<tr>
<td></td>
<td><code>string</code></td>
<td></td>
<td>• Not <code>null</code> – this is the stream where the publisher publishes publications.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• <code>null</code> – the default is <code>SYSTEM.BROKER.DEFAULT.STREAM</code>. MQ limits this string to 48 bytes.</td>
</tr>
<tr>
<td>topics</td>
<td><code>string</code></td>
<td><code>none</code></td>
<td>Use the format detailed in “Syntax for topics” on page 16.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Wildcards are not allowed.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>These are the topics on which the publisher provides information on.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>This is a required property, and generates an error if omitted.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Property</th>
<th>Values</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>topics</td>
<td><code>string</code></td>
<td><code>none</code></td>
<td>Use the format detailed in “Syntax for topics” on page 16.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>These are the topics on which the subscriber wants to receive publications.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>This is a required property, and generates an error if omitted.</td>
</tr>
<tr>
<td>anon</td>
<td><code>yes</code></td>
<td><code>no</code></td>
<td>If:</td>
</tr>
<tr>
<td></td>
<td><code>no</code></td>
<td></td>
<td>• <code>yes</code> – MQ pub/sub broker does not divulge the identity of the subscriber.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• <code>no</code> – MQ pub/sub broker divulges the identity of the subscriber.</td>
</tr>
<tr>
<td>local</td>
<td><code>yes</code></td>
<td><code>no</code></td>
<td>If:</td>
</tr>
<tr>
<td></td>
<td><code>no</code></td>
<td></td>
<td>• <code>yes</code> – the subscription is not distributed to other brokers in the network. Only publications published from this node by a publisher specifying Local are sent to this subscriber.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• <code>no</code> – the subscription is not specified in the RFH command.</td>
</tr>
</tbody>
</table>
### Property Values

<table>
<thead>
<tr>
<th>Property</th>
<th>Values</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>newPubsOnly</td>
<td>• yes</td>
<td>no</td>
<td>If:</td>
</tr>
<tr>
<td></td>
<td>• no</td>
<td></td>
<td>• yes – the broker sends this publication only to this subscriber, and retained publications that exist at registration time are not sent.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• no – the publication is not specified in the RFH command.</td>
</tr>
<tr>
<td>pubOnReqOnly</td>
<td>• yes</td>
<td>no</td>
<td>If:</td>
</tr>
<tr>
<td></td>
<td>• no</td>
<td></td>
<td>• yes – the broker only sends new publications to this subscriber, retained publications that exist at registration time are not sent.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• no – the publication is not specified in the RFH command.</td>
</tr>
<tr>
<td>inclStreamName</td>
<td>• yes</td>
<td>no</td>
<td>If:</td>
</tr>
<tr>
<td></td>
<td>• no</td>
<td></td>
<td>• yes – the broker adds the publication stream name in the MQRF header to each message that is forwarded to the subscriber.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• no – the publication is not specified in the RFH command.</td>
</tr>
<tr>
<td>informIfRet</td>
<td>• yes</td>
<td>no</td>
<td>If:</td>
</tr>
<tr>
<td></td>
<td>• no</td>
<td></td>
<td>• yes – the broker informs the subscriber if the publication is retained, by setting the MQPSPubsOptsIsRetainedPub in the MQRF header of the message sent to the subscriber.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• no – the publication is not specified in the RFH command.</td>
</tr>
<tr>
<td>dupsOk</td>
<td>• yes</td>
<td>no</td>
<td>If:</td>
</tr>
<tr>
<td></td>
<td>• no</td>
<td></td>
<td>• yes – the broker is allowed to occasionally deliver a duplicate publication to the subscriber.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• no – the publication is not specified in the RFH command.</td>
</tr>
<tr>
<td>Property</td>
<td>Values</td>
<td>Default</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------</td>
<td>-----------------------------------------</td>
<td>---------</td>
<td>-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>pubsPersistence</td>
<td>• non-persistent &lt;br&gt; • persistent &lt;br&gt; • asPublication &lt;br&gt; • asQueue</td>
<td>asQueue</td>
<td>If:&lt;br&gt; • non-persistent – the publication is placed on the subscriber queue as a nonpersistent message.&lt;br&gt; • persistent – the publication is placed on the subscriber queue as a persistent message.&lt;br&gt; • asPublication – the publication is placed on the subscriber queue with the same persistence as the original publication.&lt;br&gt; • asQueue – the publication is placed on the subscriber queue with the default persistence of the subscriber queue.</td>
</tr>
<tr>
<td>streamName</td>
<td>• null &lt;br&gt; • string</td>
<td>null</td>
<td>If: &lt;br&gt; • Not null – this is the stream where the publisher publishes publications.&lt;br&gt; • null – the subscription is identified by its traditional identity.</td>
</tr>
<tr>
<td>qmgrName</td>
<td>• null &lt;br&gt; • string</td>
<td>null</td>
<td>This is the Queue Manager used to determine the subscriber’s traditional identity. MQ limits this string to 48 bytes.</td>
</tr>
<tr>
<td>queueName</td>
<td>• null &lt;br&gt; • string</td>
<td>null</td>
<td>This is the queue used to determine the subscriber’s traditional identity. MQ limits this string to 48 bytes.</td>
</tr>
<tr>
<td>correlationAsId</td>
<td>• yes &lt;br&gt; • no &lt;br&gt; • generate</td>
<td>no</td>
<td>If:&lt;br&gt; • yes – correlationId is used as part of the subscriber’s traditional identity. You must specify correlationId, but not as 0x00.&lt;br&gt; • no – correlationId is not used as part of the subscriber’s traditional identity.&lt;br&gt; • generate – a system-generated correlationId is used as part of the subscriber’s traditional identity.</td>
</tr>
</tbody>
</table>

The properties in Table 4-20 are effective only if rhfCommand is requestUpdate.
### Table 4-20: MQSeries `msgsend` properties if `rfhCommand` is set to `requestUpdate`

<table>
<thead>
<tr>
<th>Property</th>
<th>Values</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>topics</td>
<td>string</td>
<td>none</td>
<td>Use the format detailed in “Syntax for topics” on page 16. The topic that the subscriber is requesting. Only one topic can be supplied. This is a required property, and generates an error if omitted.</td>
</tr>
<tr>
<td>streamName</td>
<td>• null</td>
<td>null</td>
<td>If:</td>
</tr>
<tr>
<td></td>
<td>• string</td>
<td></td>
<td>• Not null – this is the stream where the publisher publishes publications.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• null – the default is <code>SYSTEM.BROKER.DEFAULT.STREAM</code>.</td>
</tr>
<tr>
<td>qmgrName</td>
<td>• null</td>
<td>null</td>
<td>This is the Queue Manager name used to establish the subscriber’s traditional identity. Specify it as the same value you specified when you registered the subscriber. MQ limits this string to 48 bytes.</td>
</tr>
<tr>
<td></td>
<td>• string</td>
<td></td>
<td></td>
</tr>
<tr>
<td>queueName</td>
<td>• null</td>
<td>null</td>
<td>This is the queue used to establish the subscriber’s traditional identity. Specify it as the same value you specified when you registered the subscriber. MQ limits this string to 48 bytes.</td>
</tr>
<tr>
<td></td>
<td>• string</td>
<td></td>
<td></td>
</tr>
<tr>
<td>correlationAsId</td>
<td>• yes</td>
<td>no</td>
<td>If:</td>
</tr>
<tr>
<td></td>
<td>• no</td>
<td></td>
<td>• yes – correlationId is used as part of the subscriber’s traditional identity. You must specify correlationId, but not as 0x00.</td>
</tr>
<tr>
<td></td>
<td>• generate</td>
<td></td>
<td>• no – correlationId is not used as part of the subscriber’s traditional identity.</td>
</tr>
<tr>
<td></td>
<td>• generate</td>
<td></td>
<td>• generate – a system-generated correlationId is used as part of the subscriber’s traditional identity.</td>
</tr>
</tbody>
</table>

- 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 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_uri has the form “tcp://hostname:port”. The host name can be a name or an IP address.
- A service_provider_uri cannot have spaces.

MQSeries

The status returned by msgsend is the completion status from sending the message to the specified queue. It is not the completion status from the MQ pub/sub broker. To get the completion status from the MQ pub/sub broker, specify a replyToQueue, then send a request message or request a negativeActionReport. The MQ pub/sub broker sends a response or report MQRFH message to replyToQueue. In both cases, you must explicitly read the response or report message from the replyToQueue, and check the MQPSCompCode, MQPSReason, and MQPSReasonText properties in the received message.
When you specify `msgSegment` or `msgLastSegment`, if the application that is reading the message (by specifying `MQGMO_COMPLETE_MSG` for a non-Adaptive Server application, or `completeMsg=yes` for an Adaptive Server application), all the messages making up that logical message must be sent in a unit of work, so you must send all of the messages that need to be grouped in a single transaction.

Permissions

- You must have `messaging_role` to run `msgsend`.
msgsubscribe

Description
TIBCO JMS only – 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 msgunssubscribe, you must register the subscription with sp_msgadmin. This example registers the durable subscription "subscription_1":

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

• Once msgsubscribe is called, all messages published on the specified topic that qualify for the selector are held until msgconsume is called to read the messages. If you do not want to hold messages that arrive before you are ready to consume them, do not call msgsubscribe. Calling msgconsume without previously calling msgsubscribe starts the subscription when msgconsume is called.

• msgsubscribe starts a subscription for the client to receive messages defined by the endpoint and filter specified by subscription_name. It returns 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 4-21: SQL sessions**

<table>
<thead>
<tr>
<th>Session 1</th>
<th>Session 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>`select msgunsubscribe</td>
<td>`select msgunsubscribe</td>
</tr>
<tr>
<td>('subscription_1' WITH</td>
<td>('subscription_1' WITH</td>
</tr>
<tr>
<td>RETAIN)</td>
<td>RETAIN)</td>
</tr>
<tr>
<td>`select msgconsume</td>
<td>`select msgconsume</td>
</tr>
<tr>
<td>('subscription_1')</td>
<td>('subscription_1')</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>`select msgconsume</td>
<td>`select msgconsume</td>
</tr>
<tr>
<td>('subscription_1')</td>
<td>('subscription_1')</td>
</tr>
<tr>
<td>`select msgunsubscribe</td>
<td>`select msgunsubscribe</td>
</tr>
<tr>
<td>('subscription_1' WITH</td>
<td>('subscription_1' WITH</td>
</tr>
<tr>
<td>RETAIN)</td>
<td>RETAIN)</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

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

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

  The client consumes the message multiple times, and uses other application logic not related to messaging. 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')
  ```
msgsubscribe

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

    select msgunsubscribe('subscription_1')
msgunsubscribe

Description
TIBCO JMS only – provides a SQL interface to subscribe or unsubscribe to a topic.

Syntax
\[
\text{msg\_unsubscribe} ::= \text{msgunsubscribe} \\
\quad (\text{subscription\_name} \ [\text{with} \ (\text{\{remove} | \text{retain}\)])}
\]

Parameters
\[
\text{subscription\_name} ::= \text{basic\_character\_expression}
\]

\[\text{with(\{remove} | \text{retain}\)]
removes or retains the durable subscription from the JMS message provider.

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

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

Usage
- Before you specify a subscription with msgsubscribe or msgunsubscribe, you must register the subscription with sp_msgadmin. This example registers the durable subscription “subscription_1”:

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

- msgunsubscribe stops any current subscription for the client to the endpoint and filter specified by subscription_name. It returns a 0 if it succeeds, or 1 if it fails.

- If you specify with retain, the connection to the JMS messaging provider is terminated so that another subscription can connect, using the same subscriber client_id specified in the subscription. The durable subscriber remains defined within Adaptive Server and within the JMS message provider. If you specify with remove, the durable subscriber definition is removed from the JMS message provider. The default value is with retain.

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

```
<login>
select msgsubscribe('subscription_1')
select msgconsume('subscription_1')
```
... 
select 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 4-22: SQL sessions

<table>
<thead>
<tr>
<th>Session 1</th>
<th>Session 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>select msgunsubscribe ('subscription_1' WITH RETAIN)</td>
<td>select msgsubscribe ('subscription_1')</td>
</tr>
<tr>
<td>select msgconsume ('subscription_1')</td>
<td>select msgconsume ('subscription_1')</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>select msgconsume ('subscription_1')</td>
<td>select msgconsume ('subscription_1')</td>
</tr>
<tr>
<td>select msgunsubscribe ('subscription_1' WITH RETAIN)</td>
<td>select msgunsubscribe('subscription_1' WITH RETAIN)</td>
</tr>
</tbody>
</table>
The following example shows `msgsubscribe` used before the application logic is ready to read the messages that force the JMS client to hold messages. The application subscribes:

```sql
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:

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

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

```sql
select msgunsubscribe('subscription_1')
```
endpoint

Description
MQSeries – specifies the general syntax and processing for endpoint for WebSphere MQSeries. Individual options are described in the functions and stored procedures that accept an endpoint argument.

Note JMS endpoints are opaque to Adaptive Server, and are not inspected for correctness or validity. Instead, they are sent directly to the JMS provider.

Syntax

```
service_provider_uri ::= 
  provider_name?qmgr=qmgr_name,destination

provider_name ::= 
  local_name | full_name
  local_name ::= identifier
  full_name ::= service_provider_class:service_provider_url
  service_provider_class ::= ibm_mq
  service_provider_url ::= 
    channel_name/tcp/hostname(port)
    channel_name ::= identifier
    hostname ::= identifier
    port ::= integer

qmgr_name ::= identifier
destination ::= [remote_qmgr,]queue=queue_name

remote_qmgr ::= remote_qmgr=remote_qmgr_name
remote_qmgr_name ::= identifier
queue_name ::= identifier
```

Parameters

local_name
  is the name of a registered publisher or subscriber.

qmgr_name
  is the name of a MQSeries Queue Manager. MQ limits the length of a Queue Manager name to 48 characters (bytes).

ibm_mq
  defines the service provider class. It can be upper or lower case.

channel_name
  is the name of the MQSeries client channel. MQ limits the length of a channel name to 20 characters (bytes).

tcp
  is the transport protocol.

hostname
  is the host name of the machine where the MQSeries listener is running.
**port**

is the port number where the MQSeries listener is listening.

---

**Note** You cannot exceed 264 bytes in the combined length of `hostname(port)`.

---

**queue_name**

is the name of a MQSeries queue. MQ limits the length of a queue name to 48 characters (bytes).

**remote_qmgr_name**

is the name of the MQSeries Queue Manager. MQ limits the length of a Queue Manager name to 48 characters (bytes).

Use `remote_qmgr` when there is a remote queue. For example:

```
ibm_mq:CHANNEL2/TCP/host2(5678)?qmgr=QM2,
remote_qmgr=QM3,queue=QM3.QUEUE
```

In the example:
- QM2 – is the Queue Manager that accepts the connection on channel 'CHANNEL2'.
- QM3.QUEUE – is owned by remote Queue Manager QM3.
- QM2 – establishes a Queue Manager channel to QM3.

You must have a server-to-server channel between QM2 and QM3.

---

**Note** You must specify `qmgr`, `remote_qmgr`, and `queue_name` in that order.

---

The access to the MQ queue is made as the Adaptive Server login user. Unlike the TIBCO JMS support, you cannot specify a user name and password with the endpoint. This means that all Adaptive Server logins that are performing messaging operations must be valid MQ users. Since MQ uses the OS user identities, the Adaptive Server login must also have a user account on the machine where the MQ Queue Manager is running.

**Examples**

Sends the message, “hello world 1” to a local queue, which is already available on the Queue Manager once MQ is installed:

```
select msgsend('hello world 1',
ibm_mq:channel1/tcp/host1(5678)?qmgr=QM1,
queue=SYSTEM.DEFAULT.LOCAL.QUEUE')
```
Example 14 Sends the message, “hello world 2” to a queue:

```sql
select msgsend('hello world 2',
'ibm_mq:channel2/tcp/host2(5678)?qmgr=QM2,queue=SYSTEM.DEFAULT.QUEUE')
```

Example 15 Sends the message, “hello world 3” to a queue:

```sql
select msgsend('hello world 3',
'ibm_mq:channel2/tcp/host2(5678)?qmgr=QM2,remote_qmgr=QM3,queue=QM3.Q')
```
option_string

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

Syntax
\[\text{option_string} \,:= \text{basic_character_expression}\]
\[\text{option_string_value} \,:= \text{option_and_value} [\[,\] \text{option_and_value}]\]
\[\text{option_and_value} \,:= \text{option_name} = \text{option_value}\]
\[\text{option_name} \,:= \text{simple_identifier}\]
\[\text{option_value} \,:= \text{simple_identifier} | \text{quoted_string} | \text{integer_literal} | \text{float_literal} | \text{byte_literal} | \text{true} | \text{false} | \text{null}\]

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.

ture
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 88.
sizespec

Description

MQSeries only – message options and property values that accept a size accept the following syntax as a size specification. Message options and property values that accept a size specification accept the following syntax as a size specification for MQSeries.

Syntax

sizespec ::= integer_number [ sizespec_units ]
sizespec_units ::= { M | K }

Parameters

integer_number

is the size.

K or k

is kilobytes.

M or m

is megabytes.

sizespec_units

is the size specification in megabytes (M) or kilobytes (K), or bytes.

If you do not provide sizespec_units, the default is bytes.

Examples

Example 1 shows the size specification for 100MB:

```
-- Specify buffer length to be 100 megabytes
select msgrecv('ibm_mq:channel1/tcp/host1(5678)?
 + 'qmgr=QM1,queue=SYSTEM.DEFAULT.LOCAL.QUEUE'
 option 'bufferLength=100M')
```

Example 2 shows the size specification for 300K:

```
-- Specify buffer length to be 300 kilobytes
select msgrecv(
 'ibm_mq:channel2/tcp/host2(5678)?qmgr=QM2,remote_qmgr=QM3,queue=QM3.Q'
 option 'bufferLength=300K')
```

Example 3 MQSeries – shows the size specification for 1MB:

```
-- bufferLength specified as 1 megabyte
select msgrecv(
 'ibm_mq:channel1/tcp/host1(5678)?qmgr=QM1,queue=DEFAULT.QUEUE'
 option 'bufferLength=1M')
```

Example 4 MQSeries – shows the size specification for 10K:

```
-- bufferLength specified as 10K
select msgrecv(
 'ibm_mq:channel1/tcp/host1(5678)?qmgr=QM1,queue=DEFAULT.QUEUE'
 option 'bufferLength=10K')
```
**timespec**

**Description**
Message options and property values that accept a time interval using the `timespec` function accept the following syntax as a time specification for both MQSeries and TIBCO JMS.

**Syntax**
```
'timeout=timespec'

timespec ::= INTEGER_NUMBER [ timespec_units ]

timespec_units ::= { dd | hh | mi | ss | ms }
```

**Parameters**
- `dd` is days
- `hh` is hours
- `mi` is minutes
- `ss` is seconds
- `ms` is milliseconds

If you do not provide `timespec_units`, the default is milliseconds.

**Examples**

**Example 1** Shows the time specification for 100 days:

```
-- timeout specified as 100 days
select msgrecv('ibm_mq:channel1/tcp/host1(5678)?
+ qmgr=QM1,remote_qmgr=QM2,queue=SYSTEM.DEFAULT.LOCAL.QUEUE'
  option 'timeout=100dd')
```

**Example 2** Shows the time specification for 300 minutes:

```
-- timeout specified as 300 minutes
select msgrecv('ibm_mq:channel1/tcp/host1(5678)?
+ qmgr=QM1,queue=SYSTEM.DEFAULT.LOCAL.QUEUE'
  option 'timeout=300mi')
```

**Example 3** Shows the time specification for 1,024 milliseconds:

```
-- timeout specified as 1,024 milliseconds
select msgrecv('ibm_mq:channel1/tcp/host1(5678)?
+ qmgr=QM1,queue=SYSTEM.DEFAULT.LOCAL.QUEUE'
  option 'timeout=1000ms')
```
Example 4 MQSeries – shows the time specification for 30 seconds:

```
-- timeout specified as 30 seconds
select msgrecv('ibm_mq:channel1/tcp/host1(5678)?qmgr=QM1,queue=DEFAULT.QUEUE'
  option 'timespec=30ss')
```

Example 5 JMS – shows the time specification for 30 minutes:

```
-- timeout specified as 30 minutes
select msgrecv('tibco)_jms:tcp://localhost:7222?queue=queue.sample'
  option 'timeout=30mi')
```

See also msgconsume, msgpublish, msgrecv, msgsend
This chapter describes transactional message requirements and behavior.

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Transactional message behavior</td>
<td>137</td>
</tr>
</tbody>
</table>

### 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:

```sql
set transactional messaging [ none | simple | full ]
```
Transactional message behavior

- **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:

  ```sql
  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:

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

  In this example, `msgsend` is rolled back:

  ```sql
  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.

  ```sql
  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.
This chapter describes sample code illustrating messaging functionality that is distributed with Adaptive Server Real Time Messaging Services (RTMS).

**Topic** | **Page**
---|---
Sybase directories | 139
Using code samples with Replication Server function strings | 140
Using code samples with SQL | 140
Using code samples with Java/JDBC | 140

### 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 RTMS.
- `jdbc` – JDBC samples using RTMS.

You can find the code samples in the `SYBASE/SSYBASE_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 simply `queue_listener` on a UNIX/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 existing Adaptive Server (version 12.5.2 and earlier) or any non-Adaptive Server database into the message bus.

Using code samples with SQL

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

These samples also illustrate 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

Both the JMS- and MQSeries-related terms defined here are used throughout this document.

<table>
<thead>
<tr>
<th>Term</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Broker</td>
<td>A WebSphere MQSeries process that performs subscription resolution in a pub/sub model.</td>
</tr>
<tr>
<td>Channel</td>
<td>A WebSphere MQSeries object that is a logical communication link.</td>
</tr>
<tr>
<td>Durable subscription</td>
<td>A TIBCO JMS subscription that retains messages while the client is not connected.</td>
</tr>
<tr>
<td>JMS</td>
<td>TIBCO Java Message Service.</td>
</tr>
<tr>
<td>Messaging client</td>
<td>A TIBCO JMS program that produces or consumes messages.</td>
</tr>
<tr>
<td>MOM</td>
<td>TIBCO JMS message-oriented middleware.</td>
</tr>
<tr>
<td>MQ</td>
<td>WebSphere MQSeries Message Queue messaging system.</td>
</tr>
<tr>
<td>MQ Publish/Subscribe</td>
<td>WebSphere MQSeries publish-and-subscribe function.</td>
</tr>
<tr>
<td>MQI</td>
<td>WebSphere MQSeries Message Queue Interface programming API.</td>
</tr>
<tr>
<td>MQM</td>
<td>WebSphere MQSeries Message Queue Manager process that manages a queue.</td>
</tr>
<tr>
<td>Nondurable subscription</td>
<td>A TIBCO JMS subscription that retains messages only while the client is connected.</td>
</tr>
<tr>
<td>Queue</td>
<td>In TIBCO JMS, a domain for point-to-point messaging.</td>
</tr>
<tr>
<td></td>
<td>In WebSphere MQSeries, an object that stores sent messages.</td>
</tr>
<tr>
<td>Payload</td>
<td>A WebSphere MQSeries message body.</td>
</tr>
<tr>
<td>Publication</td>
<td>In WebSphere MQSeries, the information that is sent by a publisher.</td>
</tr>
<tr>
<td>Publisher</td>
<td>In WebSphere MQSeries, the sender in a publish/subscribe model.</td>
</tr>
</tbody>
</table>
Glossary

<table>
<thead>
<tr>
<th>Term</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>RF Header</td>
<td>The WebSphere MQSeries rules and formatting header used by MQ pub/sub. All messages sent to the MQ pub/sub broker or to the stream queue must have a RF Header. The RF Header conveys control information to the MQ pub/sub broker. In MQ pub/sub messages, the message payload contains a RF Header, followed by the application data.</td>
</tr>
<tr>
<td>RFH</td>
<td>The WebSphere MQSeries rules and formatting header; the portion of the message header that provides rules and formatting information for that message</td>
</tr>
<tr>
<td>Service provider</td>
<td>A TIBCO JMS message provider. For instance, TIBCO JMS is a service provider, called a messaging provider in this document.</td>
</tr>
<tr>
<td>Stream</td>
<td>In WebSphere MQSeries, the grouping of related MQSeries topics.</td>
</tr>
<tr>
<td>Subscriber</td>
<td>In WebSphere MQSeries, the receiver in a publish/subscribe topology.</td>
</tr>
<tr>
<td>Subscription</td>
<td>A TIBCO JMS domain for publishing or consuming one-to-many messaging.</td>
</tr>
<tr>
<td>Topic</td>
<td>In TIBCO JMS, similar to queues in, but used for one-to-many messaging.</td>
</tr>
</tbody>
</table>

In WebSphere MQSeries, the subject of a publication. WebSphere MQSeries pub/sub topics and JMS topics are different. In JMS, a topic is a pub/sub endpoint, whereas in WebSphere MQSeries pub/sub, a topic is a subject of a message.

Note: WebSphere MQSeries pub/sub topics and JMS topics are different. In JMS, a topic is a pub/sub endpoint, whereas in WebSphere MQSeries pub/sub, a topic is a subject of a message.
Index

Symbols
@@ (global variable)  32

A
Adaptive Server-specific message properties  40
ASE message types  84
ASE Replicator User’s Guide  vi
ASE_MSBODY_SCHEMA message property  40
ASE_MSGBODY message property  40
ASE_ORIGIN message property  41
ASE_RTMS_CHARSET message property  40
ASE_RTMS_VERSION message property  41
ASE_SPID message property  41
ASE_TIMESTAMP message property  41
ASE_VERSION message property  41
ASE_VERSION_FORMATS message property  41
asynchronous messaging  1

B
behavior of transactional messages  137
binary value of datatypes  84
body of message  4
broker command queue in MQSeries  4
broker, defined  141
built-ins. See functions.
byte message type  84
byte ordering  84

c
channel defined  141
code samples
  using with Java/JDBC  140
  using with Replication Server function strings

d
datatypes, binary value of  84
descriptions
  broker  141
  channels  141
  endpoint syntax segment  130
  MQSeries  11
  msgconsume function  57
  msgheader XML documents  38
  msgpropcount function  60
  msgproperties XML documents  38, 39
  msgproplist function  61
  msgpropname function  63
  msgproptype function  64
  msgpropvalue function  67
  msgpublish function  69
  msgrecv function  73
  msgsend function  88
  msgsubscribe function  124
  msgunsubscribe function  127
  option_string syntax segment  133
  queues  141
  sizespec syntax segment  134
  sp_engine stored procedure  44
  sp_msgadmin stored procedure  48
  timespec syntax segment  135
  XML documents  38
directories
  functionstring  139
Index

jdbc 139
sql 139
double keywords, new 41
durable subscriptions 3
defined 141

E
EJB Server User’s Guide vi
epoint syntax segment 130–132
described 130
described 130
described 130
described 130

described 130
described 130
described 130
described 130

described 130
described 130
described 130
described 130

described 130
described 130
described 130
described 130

Error Messages and Troubleshooting Guide vi
epoint syntax segment 131
global variables 37
msgconsume function 58
msgheader XML documents 39
msgpropcount function 60
msgproplist function 61
msgpropname function 63
msgproptype function 64
msgpropvalue function 67
msgpublish function 70
msgrecv function 74
msgsend function 92–96
msgsubscribe function 124
msgunsubscribe function 127
sizespec syntax segment 134
sp_engine stored procedure 45
sp_msagadmin stored procedure 52
timespec syntax segment 135
XML documents 39

F
filenames, different on different platforms 139
font conventions xi
Full-Text Search Specialty Data Store User’s Guide vii
functions
described 42
list of 42
msgconsume 57–59
msgpropcount 60
msgproplist 61–62
msgpropname 63
msgproptype 64–66
msgpropvalue 67–68
msgpublish 69–72
msgrecv 73–75
msgsend 88–123
msgsubscribe 124
msgunsubscribe 127
rtrim, for removing trailing blanks 38
functionstring subdirectory in $SYBASE directory 139

global variables
@@msgcorrelation 32
@@msgcreplyqmgr 36
@@msgheader 32
@@msgid 35
@@msgmsgschema 36
@@msgproperties 35
@@msgreplytoinfo 36
@@msgstatus 36
@@msgstatusinfo 37
@@msgtimestamp 37
described 37
setting 32
usages 38

Glossary vii

H
global variables
help, for installation or feature xii
Historical Server User’s Guide vii

I
Installation Guide vi
installing
RTMS 27
Index

J
Java Message Service. See JMS.
Java/JDBC, using code samples with 140
jConnect for JDBC Programmer's Reference vii
jdbc subdirectory in $SYBASE directory 139
JMS
defined 141
message properties 4
queue description 3
queue, messages read from 8
reference documents v
URL for v
Job Scheduler User's Guide vii

K
keywords, new, double and triple 41

M
message
body 4
bus, TIBCO 2
formats 4
headers 4
interface, preview of 9
properties in JMS 4
properties in MQSeries 5
properties, working with 8
read from JMS queue 8
selectors in MQSeries 5
selectors in JMS 4
sending with Transact SQL applications 7
message body 4
message filters for using msgrecv function 86
message formats 4
message headers 4
message message type 84
message properties 8
    Adaptive Server-specific 40
    Adaptive Server-specific table 40
    ASE_MSBODY_SCHEMA 40
    ASE_MSGBODY 40
    ASE_ORIGIN 41
    ASE_RTMS_CHARSET 40
    ASE_RTMS_VERSION 41
    ASE_SPID 41
    ASE_TIMESTAMP 41
    ASE_VERSION 41
    ASE_VERSION_FORMATS 41
    JMS, in 4
    MQSeries, in 5
message receivers in JMS 3
message senders in JMS 3
message types
    binary 7
    supported in msgconsume 59
    text 7
message, transactional behavior of 137
message-oriented middleware (MOM) 141
message-related global variables 32
messages
    publishing and consuming from a topic 8
    sending and receiving from a queue 7
messaging
    client 141
    concepts 1
    models 3
messaging global variables
    @@msgcorrelation 32
    @@msgheader 32
    @@msgid 35
    @@msgmsgchema 36
    @@msgproperties 35
    @@msgreplyqmgr 36
    @@msgreplyto, format 38
    @@msgreplytoinfo 36
    @@msgstatus 36
    @@msgstatusinfo 37
    @@msgtimestamp 37
    char datatypes 38
messaging models
    JMS-defined 3
    MQSeries 3
    MQSeries publish and subscribe 4
    MQSeries-defined 3
    point-to-point 2
    publish and subscribe 2
messaging provider 2
    creating, deleting, and accessing queues and topics
Index

messaging systems, asynchronous 1
models, messaging 3
MOM. See message-oriented middleware
Monitor Client Library Programmer’s Guide vii
Monitor Server User’s Guide vii
MQSeries
broker command queue 4
message properties 5
messaging models 3
msgrecv function, usage for 85
overview 11
publish and subscribe messaging model 4
RF headers 5
URL v
msgconsume function 57–59
calling, results of 59
described 57
examples 58
message types supported 59
parameters 57
syntax 57
unsupported message datatypes 59
usage 59
@@msgcorrelation messaging global variable 32
@@msgheader messaging global variable 32
msgheader XML document 38–40
described 38
examples 39
syntax 39
usage 40
@@msgid messaging global variable 35
msgpropcount function 60
described 60
examples 60
parameters 60
syntax 60
@@msgproperties messaging global variable 35
msgproperties XML document 38–40
described 38, 39
syntax 39
usage 40
msgproplist function 61–62
described 61
examples 61
parameters 61
msgpropname function 63
described 63
examples 63
parameters 63
syntax 63
msgpropvalue function 64–66
described 64
examples 64
parameters 64
syntax 64
usage 65
msgpropvalue function 67–68
described 67
examples 67
parameters 67
syntax 67
msgpublish function 69–72
described 69
examples 70
parameters 69
syntax 69
usage 70
msgrecv function 73–87
described 73
examples 74
message filters 86
parameters 73
permissions 87
syntax 73
usage 84
usage for MQSeries 85
@@msgreplyqmgr messaging global variable 36
@@msgreplytoinfo messaging global variable 36
@@msgschema messaging global variable 36
msgsend function 88–123
behavior in a transactions 137
described 88
examples 92–96
parameters 88–92
permissions 123
syntax 88
usage 96–123
@@msgstatus messaging global variable 36
@@msgstatusinfo messaging global variable 37
Index

msgsubscribe function 124
  described 124
  examples 124
  parameters 124
  syntax 124
  usage 124
@@msgtimestamp messaging global variable 37
msgunsubscribe function 127
  described 127
  examples 127
  parameters 127
  syntax 127
  usage 127

N
  non-durable subscriptions 3, 141

O
  option strings 42
  option_string syntax segment 133
    described 133
    parameters 133
    syntax 133
    usage 133

P
  parameters
    endpoint syntax segment 130
    msgconsume function 57
    msgpropcount function 60
    msgproplist function 61
    msgpropname function 63
    msgpropotype function 64
    msgpropvalue function 67
    msgpublish function 69
    msgrecv function 73
    msgsend function 88–92
    msgsubscribe function 124
    msgunsubscribe function 127
    option_string syntax segment 133

sizespec syntax segment 134
sp_engine stored procedure 44
sp_msgadmin stored procedure 49
timespec syntax segment 135
Performance and Tuning Guide vii
  performing messaging operations described 2
  permissions
    msgrecv function 87
    msgsend function 123
    sp_engine stored procedure 47
    sp_msgadmin stored procedure 55
  point-to-point messaging models 2
    JMS 3
    MQSeries 3
  point-to-point queues
    JMS 3
    MQSeries 3
  preview, examples 9
  previewing message interface 9
  provider, messaging 2
  publish and subscribe
    messaging model 2
  publish-and-subscribe
    JMS messaging model 3
    MQ messaging model 3

Q
  queues
    defined 141
    for one-to-one messaging 141
  sending and receiving messages from 7
  queues and topics, creating, deleting, accessing 42
Quick Reference Guide vii

R
  Real Time Messaging Services. See RTMS.
  receiving messages 7
Reference Manual vii
  referenced documents v
  related documents v
  ASE Replicator User’s Guide vi
  Component Integration Services User’s Guide vi

Messaging Services User’s Guide

147
Index

Configuration Guide vi
EJB Server User’s Guide vi
Error Messages and Troubleshooting Guide vi
Full-Text Search Specialty Data Store User’s Guide vii
Glossary vii
Historical Server User’s Guide vii
Installation Guide vi
jConnect for JDBC Programmer’s Reference vii
Job Scheduler User’s Guide vii
Monitor Client Library Programmer’s Guide vii
Monitor Server User’s Guide vii
Performance and Tuning Guide vii
Quick Reference Guide vii
Reference Manual vii
System Administration Guide viii
System Tables Diagram viii
Transact-SQL User’s Guide viii
Using Adaptive Server Distributed Transaction Management Features viii
Using Sybase Failover in a High Availability System viii
Utility Guide viii
Web Services User’s Guide viii
What’s New in Adaptive Server Enterprise? vii
XA Interface Integration Guide for CICS, Encina, and TUXEDO viii
XML Services in Adaptive Server Enterprise viii
release bulletin vi
RepConnector ii
Replication Server code samples 140
requeue, using for incorrect message types 84
RF headers 5
RFH. See RF headers.
RTMS
functions, for managing and administering 42
installing 27
RTMS, configuring 27
rtrim, function 38
rules and formatting headers. See RF headers.

S
sample code
overview 139
sybase directories 139
samples 139
sending messages 7
service provider 142
set transactional messaging command 137
sizespec syntax segment 134
described 134
examples 134
parameters 134
syntax 134
sp_engine stored procedure 44–47
described 44
examples 45
parameters 44
permissions 47
syntax 44
usage 46
sp_msgadmin and MQSeries 42
sp_msgadmin stored procedure 48–56
described 48
examples 52
parameters 49
permissions 55
syntax 48
usage 54
SQL
commands in a transaction 137
functions described 42
functions with message properties 8
using code samples with 140
sql subdirectory in $SYBASE directory 139
stored procedures
list of 42
sp_engine 44–47
sp_msgadmin 48–56
style conventions x
subscriptions
defined 142
durable 3
nondurable 3
support contracts with Sybase xii
Sybase Technical Support xii
syntax
endpoint syntax segment 130
msgconsume function 57
msgheader XML documents 39
Index

W
Web Services User’s Guide viii
What’s New in Adaptive Server Enterprise? vi

X
XA Interface Integration Guide for CICS, Encina, and TUXEDO viii
XML documents
  description 38
  examples 39
  msgheader 38–40
  msgproperties 38–40
  usage 40
XML Services in Adaptive Server Enterprise viii