

TIBCO Enterprise Message Service™

User's Guide

*Software Release 8.0
June 2013*

Important Information

SOME TIBCO SOFTWARE EMBEDS OR BUNDLES OTHER TIBCO SOFTWARE. USE OF SUCH EMBEDDED OR BUNDLED TIBCO SOFTWARE IS SOLELY TO ENABLE THE FUNCTIONALITY (OR PROVIDE LIMITED ADD-ON FUNCTIONALITY) OF THE LICENSED TIBCO SOFTWARE. THE EMBEDDED OR BUNDLED SOFTWARE IS NOT LICENSED TO BE USED OR ACCESSED BY ANY OTHER TIBCO SOFTWARE OR FOR ANY OTHER PURPOSE.

USE OF TIBCO SOFTWARE AND THIS DOCUMENT IS SUBJECT TO THE TERMS AND CONDITIONS OF A LICENSE AGREEMENT FOUND IN EITHER A SEPARATELY EXECUTED SOFTWARE LICENSE AGREEMENT, OR, IF THERE IS NO SUCH SEPARATE AGREEMENT, THE CLICKWRAP END USER LICENSE AGREEMENT WHICH IS DISPLAYED DURING DOWNLOAD OR INSTALLATION OF THE SOFTWARE (AND WHICH IS DUPLICATED IN THE LICENSE FILE) OR IF THERE IS NO SUCH SOFTWARE LICENSE AGREEMENT OR CLICKWRAP END USER LICENSE AGREEMENT, THE LICENSE(S) LOCATED IN THE "LICENSE" FILE(S) OF THE SOFTWARE. USE OF THIS DOCUMENT IS SUBJECT TO THOSE TERMS AND CONDITIONS, AND YOUR USE HEREOF SHALL CONSTITUTE ACCEPTANCE OF AND AN AGREEMENT TO BE BOUND BY THE SAME.

This document contains confidential information that is subject to U.S. and international copyright laws and treaties. No part of this document may be reproduced in any form without the written authorization of TIBCO Software Inc.

TIBCO, The Power of Now, TIB, Information Bus, TIBCO Enterprise Message Service, TIBCO Rendezvous, TIBCO Enterprise, TIBCO SmartSockets, TIBCO ActiveMatrix BusinessWorks, and TIBCO Hawk are either registered trademarks or trademarks of TIBCO Software Inc. in the United States and/or other countries.

Enterprise Java Beans (EJB), Java 2 Platform Enterprise Edition (J2EE), and all Java-based trademarks and logos are trademarks or registered trademarks of Oracle Corporation in the U.S. and other countries.

All other product and company names and marks mentioned in this document are the property of their respective owners and are mentioned for identification purposes only.

THIS SOFTWARE MAY BE AVAILABLE ON MULTIPLE OPERATING SYSTEMS. HOWEVER, NOT ALL OPERATING SYSTEM PLATFORMS FOR A SPECIFIC SOFTWARE VERSION ARE RELEASED AT THE SAME TIME. SEE THE README FILE FOR THE AVAILABILITY OF THIS SOFTWARE VERSION ON A SPECIFIC OPERATING SYSTEM PLATFORM.

THIS DOCUMENT IS PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT.

THIS DOCUMENT COULD INCLUDE TECHNICAL INACCURACIES OR TYPOGRAPHICAL ERRORS. CHANGES ARE PERIODICALLY ADDED TO THE INFORMATION HEREIN; THESE CHANGES WILL BE INCORPORATED IN NEW EDITIONS OF THIS DOCUMENT. TIBCO SOFTWARE INC. MAY MAKE IMPROVEMENTS AND/OR CHANGES IN THE PRODUCT(S) AND/OR THE PROGRAM(S) DESCRIBED IN THIS DOCUMENT AT ANY TIME.

THE CONTENTS OF THIS DOCUMENT MAY BE MODIFIED AND/OR QUALIFIED, DIRECTLY OR INDIRECTLY, BY OTHER DOCUMENTATION WHICH ACCOMPANIES THIS SOFTWARE, INCLUDING BUT NOT LIMITED TO ANY RELEASE NOTES AND "READ ME" FILES.

Copyright © 1997-2013 TIBCO Software Inc. ALL RIGHTS RESERVED.

TIBCO Software Inc. Confidential Information

Contents

Figures	xix
Tables	xxi
Preface	xxv
Changes from the Previous Release of this Guide	xxvi
Feature Enhancements	xxvi
Changes in Functionality	xxvi
Related Documentation	xxviii
TIBCO Enterprise Message Service Documentation	xxviii
Other TIBCO Product Documentation	xxviii
Third Party Documentation	xxix
Typographical Conventions	xxx
Connecting with TIBCO Resources	xxxiii
How to Join TIBCOCommunity	xxxiii
How to Access TIBCO Documentation	xxxiii
How to Contact TIBCO Support	xxxiii
Chapter 1 Overview	1
JMS Overview	2
JMS Message Models	3
Point-to-Point	3
Publish and Subscribe	4
Multicast	6
EMS Destination Features	8
Client APIs	10
Sample Code	10
TIBCO Rendezvous Java Applications	10
Administration	11
Administering the Server	11
User and Group Management	12
Using TIBCO Hawk	12
Security	13
Fault Tolerance	13
Routing	14

Integrating With Third-Party Products	14
Transaction Support	14
Chapter 2 Messages	15
EMS Extensions to JMS Messages	16
JMS Message Structure	17
JMS Message Header Fields	17
EMS Message Properties	19
JMS Message Bodies	22
Maximum Message Size	23
Message Priority	24
Message Delivery Modes	25
PERSISTENT	25
NON_PERSISTENT	25
RELIABLE_DELIVERY	26
How EMS Manages Persistent Messages	27
Persistent Messages Sent to Queues	27
Persistent Messages Published to Topics	27
Persistent Messages and Synchronous File Storage	28
Store Messages in Multiple Stores	30
Store Types	30
Default Store Files	31
Configuring Multiple Stores	32
Understanding mstore Intervals	33
Character Encoding in Messages	36
Supported Character Encodings	37
Sending Messages	37
Message Compression	38
About Message Compression	38
Setting Message Compression	38
Message Acknowledgement	39
NO_ACKNOWLEDGE	40
EXPLICIT_CLIENT_ACKNOWLEDGE	40
EXPLICIT_CLIENT_DUPS_OK_ACKNOWLEDGE	41
Message Selectors	42
Data Type Conversion	45
Sending Messages Synchronously and Asynchronously	46
Sending Synchronously	46
Sending Asynchronously	47
Receiving Messages Synchronously and Asynchronously	49

Chapter 3 Destinations	51
Destination Overview	52
Destination Names	53
Static Destinations	54
Dynamic Destinations	54
Temporary Destinations	54
Destination Bridges	55
Destination Name Syntax	56
Examples	57
Destination Properties	58
channel	59
exclusive	59
expiration	60
export	61
flowControl	61
global	62
import	62
maxbytes	63
maxmsgs	64
maxRedelivery	64
overflowPolicy	65
prefetch	68
redeliveryDelay	70
secure	71
sender_name	72
sender_name_enforced	72
store	73
trace	74
Creating and Modifying Destinations	75
Creating Secure Destinations	76
Wildcards	77
Wildcards * and >	77
Overlapping Wildcards and Disjoint Properties	77
Wildcards in Topics	78
Wildcards in Queues	78
Wildcards and Multicast	78
Wildcards and Dynamically Created Destinations	79
Inheritance	80
Inheritance of Properties	80
Inheritance of Permissions	81
Destination Bridges	82
Creating a Bridge	84
Access Control and Bridges	85

Transactions	85
Flow Control	87
Enabling Flow Control	87
Enforcing Flow Control	87
Multicast and Flow Control	88
Routes and Flow Control	88
Destination Bridges and Flow Control	89
Flow Control, Threads and Deadlock	89
Delivery Delay	91
Chapter 4 Getting Started	93
About the Sample Clients	94
Compiling the Sample Java Clients	95
Creating Users with the EMS Administration Tool	96
Start the EMS Server and EMS Administration Tool	96
Create Users	97
Point-to-Point Messaging Example	98
Create a Queue	98
Start the Sender and Receiver Clients	98
Publish and Subscribe Messaging Example	99
Create a Topic	99
Start the Subscriber Clients	99
Start the Publisher Client and Send Messages	100
Create a Secure Topic	101
Create a Durable Subscriber	103
Multicast Messaging Example	104
Stop the EMS Server	104
Enable the EMS Server for Multicast	104
Create a Multicast Channel	104
Start the EMS Server	105
Enable a Topic for Multicast	105
Start the Multicast Daemon	105
Start the Subscriber Client	105
Start the Publisher Client and Send Messages	106
Chapter 5 Running the EMS Server	107
Starting and Stopping the EMS Server	108
Starting the EMS Server Using the Default Configuration	108
Starting the EMS Server Using JSON Configuration	108
Starting the EMS Server Using Options	109
Stopping the EMS Server	111
Running the EMS Server as a Windows Service	112

emsntsrcg	112
Error Recovery Policy	115
Security Considerations	116
Secure Environment	116
Destination Security	116
Authorization Parameter	116
Admin Password	117
Connection Security	117
Communication Security	118
Sources of Authentication Data	118
Timestamp	118
Passwords	119
Audit Trace Logs	119
How EMS Manages Access to Shared Store Files	120
Performance Tuning	121
Setting Thread Affinity for Increased Throughput	121
Determining Core Allocation	121
Network I/O Connections	122
Other Considerations	122
Chapter 6 Using the EMS Administration Tool	123
Starting the EMS Administration Tool	124
When You First Start tibemsadmin	126
Naming Conventions	127
Name Length Limitations	127
Command Listing	128
add member	128
addprop factory	128
addprop queue	128
addprop route	129
addprop topic	129
autocommit	129
commit	129
compact	130
connect	130
create bridge	131
create durable	131
create factory	131
create group	131
create jndiname	131
create queue	132
create route	132
create rvcmlistener	132

create topic	133
create user	133
delete all	133
delete bridge	133
delete connection	134
delete durable	134
delete factory	134
delete group	134
delete jndiname	134
delete message	134
delete queue	135
delete route	135
delete rvcmlistener	135
delete topic	135
delete user	135
disconnect	136
echo	136
exit	136
grant queue	136
grant topic	137
grant admin	138
help	138
info	138
jaci clear	138
jaci resetstats	138
jaci showstats	138
purge all queues	139
purge all topics	139
purge durable	139
purge queue	139
purge topic	139
remove member	139
removeprop factory	140
removeprop queue	140
removeprop route	140
removeprop topic	140
resume route	140
revoke admin	140
revoke queue	141
revoke topic	141
rotatelog	142
set password	142
set server	143
setprop factory	148
setprop queue	148

setprop route	149
setprop topic	149
show bridge	149
show bridges	150
show channel	150
show channels	151
show config.	152
show consumer.	152
show consumers.	152
show connections.	156
show db	159
show durable	159
show durables.	160
show factory	161
show factories.	161
show jndiname	162
show jndinames	162
show group	162
show groups	162
show members	162
show message	163
show messages	163
show parents	163
show queue	164
show queues.	165
show route	166
show routes	167
show rvcmltransportledger	167
show rvcmllisteners.	168
show server	168
show stat.	168
show store	169
show stores.	171
show topic.	171
show topics.	173
show subscriptions	175
show transaction.	176
show transactions.	178
show transport	179
show transports	179
show user	180
show users	180
showacl admin	180
showacl group.	180
showacl queue	180

showacl topic	181
showacl user	181
shutdown	181
suspend route	181
time	182
timeout	182
transaction commit	182
transaction rollback	182
updatecrl	182
whoami	183
Chapter 7 Using the Configuration Files	185
Location of Configuration Files	186
Mechanics of Configuration	186
tibemsd.conf	187
Global System Parameters	199
Storage File Parameters	206
Connection and Memory Parameters	207
Detecting Network Connection Failure Parameters	211
Fault Tolerance Parameters	213
Message Tracking Parameters	217
Multicast Parameters	217
TIBCO Rendezvous Parameters	219
TIBCO SmartSockets Parameters	219
Tracing and Log File Parameters	220
Statistic Gathering Parameters	222
SSL Server Parameters	224
LDAP Parameters	228
Extensible Security Parameters	235
JVM Parameters	237
Using Other Configuration Files	238
acl.conf	239
bridges.conf	240
channels.conf	241
durables.conf	244
factories.conf	245
groups.conf	249
jaas.conf	250
queues.conf	250
routes.conf	251
stores.conf	253
tibrvcm.conf	257
topics.conf	257

transports.conf	258
users.conf	262
Chapter 8 Authentication and Permissions	265
EMS Access Control	266
Administrator Permissions	267
Predefined Administrative User and Group	267
Granting and Revoking Administration Permissions	268
Enforcement of Administrator Permissions	269
Global Administrator Permissions	269
Destination-Level Permissions	272
Protection Permissions	273
Enabling Access Control	275
Server Control	275
Destination Control	276
Users and Groups	277
Users	278
Groups	279
Configuring an External Directory	279
User Permissions	283
Example of Setting User Permissions	284
Inheritance of User Permissions	284
Revoking User Permissions	285
When Permissions Are Checked	286
Example of Permission Checking	287
Chapter 9 Extensible Security	289
Overview of Extensible Security	290
Extensible Authentication	292
Enabling Extensible Authentication	292
Writing an Authentication Module	293
Extensible Permissions	295
Cached Permissions	295
How Permissions are Granted	296
Implications of Wildcards on Permissions	298
Enabling Extensible Permissions	299
Writing a Permissions Module	300
The JVM in the EMS Server	302
Enabling the JVM	302

Chapter 10 Using Database Stores	303
Database Store Overview	304
Configuring Database Stores	305
Configuration in tibemsd.conf	306
Configuration in stores.conf	307
Configuration for the Oracle RAC Database	311
EMS Schema Export Tool	312
Chapter 11 Developing an EMS Client Application	317
JMS Specification	318
JMS 2.0 Specification	318
JMS 1.1 Specification	319
JMS 1.0.2b Specification	319
Sample Clients	320
Programmer Checklists	321
Java Programmer's Checklist	321
C Programmer's Checklist	322
C# Programmer's Checklist	328
Connection Factories	332
Looking up Connection Factories	332
Dynamically Creating Connection Factories	332
Setting Connection Attempts, Timeout and Delay Parameters	333
Connecting to the EMS Server	335
Starting, Stopping and Closing a Connection	336
Creating a Session	337
Setting an Exception Listener	338
Dynamically Creating Topics and Queues	340
Creating a Message Producer	342
Configuring a Message Producer	343
Creating a Completion Listener for Asynchronous Sending	344
Creating a Message Consumer	346
Creating a Message Listener for Asynchronous Message Consumption	348
Working with Messages	352
Creating Messages	352
Setting and Getting Message Properties	353
Sending Messages	354
Receiving Messages	355
Chapter 12 Using the EMS Implementation of JNDI	359
Creating and Modifying Administered Objects in EMS	360

Creating Connection Factories for Secure Connections	360
Creating Connection Factories for Fault-Tolerant Connections	361
Looking up Administered Objects Stored in EMS	362
Looking Up Objects Using Full URL Names	364
Performing Secure Lookups	365
Performing Fault-Tolerant Lookups	366
Chapter 13 Using Multicast	369
Overview of Multicast	370
When to Use Multicast	371
Requirements	373
Configuring Multicast	374
Configuring Multicast Dynamically	374
Configuring the Multicast Daemon	375
Controlling Access to Multicast-Enabled Topics	378
Running Multicast	379
Starting the Multicast Daemon	379
Creating a Multicast Consumer	379
Monitoring and Statistics	380
Monitoring	380
Statistics	380
Chapter 14 Multicast Deployment and Troubleshooting	381
Deployment Considerations	382
Connectivity	382
Restricting Multicast Traffic	384
Managing Bandwidth	384
Walking Through a Multicast Deployment	388
Step 1: Design the Multicast Network Architecture	388
Step 2: Install and Set Up EMS	390
Step 3: Determine Network and Application Capabilities	393
Troubleshooting EMS Multicast	397
Troubleshooting Tips	397
Application and Multicast Daemon Errors and Warnings	399
Server Errors	400
Chapter 15 Working With TIBCO Rendezvous	401
Overview	402
Message Translation	402
Configuration	402
Configuring Transports for Rendezvous	404

Transport Definitions	405
Topics	410
Import Only when Subscribers Exist	410
Wildcards	410
Certified Messages	411
Queues	412
Configuration	412
Import—Start and Stop	412
Wildcards	412
Import Issues	414
Field Identifiers	414
JMSDestination	414
JMSReplyTo	414
JMSExpiration	414
JMSTimestamp	415
Guaranteed Delivery	415
Export Issues	416
JMSReplyTo	416
Certified Messages	416
Guaranteed Delivery	416
Message Translation	417
JMS Header Fields	417
JMS Property Fields	418
Message Body	419
Data Types	421
Pure Java Rendezvous Programs	423
Chapter 16 Working With TIBCO SmartSockets	425
Overview	426
Message Translation	426
Configuration	426
Starting the Servers	427
Configuring Transports for SmartSockets	427
Transport Definitions	428
Destination Name—Syntax and Semantics	432
Topics	433
Import Only when Subscribers Exist	433
Wildcards	434
Queues	434
Configuration	434
Import—Start and Stop	434
Wildcards	435

Import Issues	436
Import Destination Names Must be Unique	436
JMSReplyTo	436
Guaranteed Delivery	436
Export Issues	437
JMSReplyTo	437
Wildcard Subscriptions	437
Guaranteed Delivery	437
Message Translation	438
JMS Header Fields	438
JMS Property Fields	438
SmartSockets Message Properties	439
Message Body	440
Data Types	442
Destination Names	443
Chapter 17 Monitoring Server Activity	445
Log Files and Tracing	446
Configuring the Log File	446
Tracing on the Server	447
Message Tracing	452
Enabling Message Tracing for a Destination	452
Enabling Message Tracing on a Message	453
Monitoring Server Events	454
System Monitor Topics	454
Monitoring Messages	454
Viewing Monitor Topics	457
Performance Implications of Monitor Topics	458
Working with Server Statistics	460
Overall Server Statistics	460
Enabling Statistic Gathering	461
Displaying Statistics	463
Chapter 18 Using the SSL Protocol	465
SSL Support in TIBCO Enterprise Message Service	466
Digital Certificates	467
Digital Certificate File Formats	468
Private Key Formats	468
File Names for Certificates and Keys	469
Configuring SSL in the Server	471
SSL Parameters	471
Command Line Options	471

Configuring SSL in EMS Clients	472
Client Digital Certificates	472
Configuring SSL	473
Specifying Cipher Suites	476
Syntax for Cipher Suites	476
Supported Cipher Suites	478
SSL Authentication Only	482
Enabling FIPS Compliance	483
Enabling the EMS Server	483
Enabling EMS Clients	484
Chapter 19 Fault Tolerance	485
Fault Tolerance Overview	486
Shared State	486
Unshared State Failover	486
Configuration Files	487
Shared State Failover Process	488
Detection	488
Response	488
Role Reversal	489
Client Transfer	489
Message Redelivery	490
Heartbeat Parameters	491
Unshared State Failover Process	492
Dual State Failover	493
Shared State	495
Implementing Shared State	495
Messages Stored in Shared State	497
Storage Files	497
Storage Parameters	498
Configuring Fault-Tolerant Servers	499
Shared State	499
Unshared State	501
Configuring Fault Tolerance in Central Administration	502
Configuring Clients for Fault-Tolerant Connections	504
Specifying More Than Two URLs	504
Setting Reconnection Failure Parameters	505
Configuring Clients for Unshared State Connections	507
Include the Unshared State Library	507
Create an Unshared State Connection Factory	507
Specify Server URLs	508

Chapter 20 Working With Routes	509
Overview of Routing	510
Route	511
Basic Operation	511
Global Destinations	512
Unique Routing Path	512
Zone	514
Basic Operation	514
Eliminating Redundant Paths with a One-Hop Zone	514
Overlapping Zones	515
Active and Passive Routes	517
Configuring Routes and Zones	518
Routes to Fault-Tolerant Servers	519
Routing and SSL	519
Routed Topic Messages	523
Propagating Registered Interest	523
Selectors for Routing Topic Messages	525
Routed Queues	528
Routing and Authorization	531
Appendix A Monitor Messages	533
Description of Monitor Topics	534
Description of Topic Message Properties	537
Appendix B Error and Status Messages	547
Error and Status Messages	548
Index	613

Figures

Figure 1	Message Delivery	3
Figure 2	Point-to-point messages	4
Figure 3	Publish and subscribe messages	5
Figure 4	Multicast messages	7
Figure 5	Persistent Message Delivery	25
Figure 6	Non-Persistent Message Delivery	26
Figure 7	Reliable Message Delivery	26
Figure 8	Persistent Messages Sent to a Queue	27
Figure 9	Persistent Messages Published to a Topic	28
Figure 10	Message Delivery and Acknowledgement	39
Figure 11	Bridging a topic to a queue	83
Figure 12	Bridging a topic to multiple destinations	83
Figure 13	Bridging a queue to multiple destinations	84
Figure 14	Flow Control Deadlock across Two Threads	90
Figure 15	Users, groups, and permissions	277
Figure 16	Methods for authenticating users and checking permissions	291
Figure 17	The Permissions Decision Tree	297
Figure 18	Multicast message consumer creation	370
Figure 19	The benefits of multicast	372
Figure 20	Sample Multicast Deployment Architecture	389
Figure 21	Rendezvous Transports in the EMS Server	402
Figure 22	SmartSockets Transports in the EMS Server	426
Figure 23	Primary and Backup Servers with Shared State	486
Figure 24	Current and Second Servers with Unshared State	487
Figure 25	Failed Primary Server	488
Figure 26	Recovered Server Becomes Backup	489
Figure 27	Unshared State Failover	492
Figure 28	Dual State Failover Process	494

Figure 29 Routes: bidirectionality and corresponding destinations 511

Figure 30 Routes: global destinations 512

Figure 31 Routes: Unique Path 513

Figure 32 Zones: multi-hop 514

Figure 33 Zones: one-hop 515

Figure 34 Zones: overlap. 516

Figure 35 Routing: Propagating Subscribers. 523

Figure 36 Routing: Topic Selectors, example 525

Figure 37 Routing: Queues 528

Figure 38 Routing: Authorization 531

Tables

Table 1	General Typographical Conventions	xxx
Table 2	Syntax Typographical Conventions	xxxi
Table 3	Summary of administration features.	11
Table 4	JMS Message Headers	17
Table 5	Summary of message properties	19
Table 6	JMS Message Types	22
Table 7	Data Type Conversion	45
Table 8	Destination Overview	52
Table 9	Characters with Special Meaning in Destination Names	56
Table 10	Valid Destination Name Examples	57
Table 11	Invalid Destination Name Examples.	57
Table 12	Destination properties	58
Table 13	Prefetch	68
Table 14	tibemsd Options	110
Table 15	tibemsaadmin Options	124
Table 16	set server — parameters	143
Table 17	show channel — description of output fields	150
Table 18	show consumers — description of output fields.	153
Table 19	show connections — type parameter.	156
Table 20	show connections — description of output fields	157
Table 21	show durable — table Information	159
Table 22	show durables — table Information	161
Table 23	show queue — table Information	164
Table 24	show queues — table Information	165
Table 25	show routes — table Information	167
Table 26	show store — table Information	169
Table 27	show topic — table Information	171
Table 28	show topics — table information	174

Table 29 show subscriptions — table information 176

Table 30 show transactions — table information 177

Table 31 show transactions — table information 179

Table 32 tibemsd.conf Parameters 187

Table 33 Configuration Files 238

Table 34 ACL Parameters 239

Table 35 Bridge Parameters 240

Table 36 Channel Parameters 241

Table 37 Durable Subscriber Parameters 244

Table 38 Connection Factory Parameters 245

Table 39 Group Parameters 249

Table 40 Route Parameters 251

Table 41 Store File Parameters 253

Table 42 RVCM Listener Parameters 257

Table 43 Transport Parameters 258

Table 44 User Parameters 262

Table 45 Global administrator permissions 269

Table 46 Destination-level administration permissions 272

Table 47 Default configuration for popular LDAP servers 281

Table 48 Queue Permission 283

Table 49 Topic Permission 284

Table 50 Database Store File Parameters 308

Table 51 EMS Schema Export Tool options 313

Table 52 Linker Flags for 32-Bit UNIX 324

Table 53 Linker Flags for 64-Bit UNIX 324

Table 54 Dynamic Library Files for Microsoft Windows 325

Table 55 Static Library Files for Microsoft Windows 326

Table 56 Shareable Image Library Files for OpenVMS 327

Table 57 Static Library Files for OpenVMS 327

Table 58 EMS Assembly DLL 328

Table 59 EMS Policy Files 329

Table 60 .NET Feature Support 330

Table 61	tibemsmcd Options	376
Table 62	Rendezvous: Transport Parameters	405
Table 63	Rendezvous: Mapping JMS Header Fields to RV Datatypes.....	417
Table 64	Rendezvous Mapping Message Properties	418
Table 65	Rendezvous: Mapping Message Types (Import)	419
Table 66	Rendezvous: Mapping Message Types (Export)	420
Table 67	Rendezvous: Mapping Data Types	421
Table 68	SmartSockets: Transport Parameters	428
Table 69	SmartSockets Mapping Message Properties (Import & Export).....	439
Table 70	SmartSockets: Mapping Message Types (Export).....	441
Table 71	SmartSockets: Mapping Data Types	442
Table 72	Server Tracing Options	448
Table 73	Message monitoring qualifiers	455
Table 74	File types	469
Table 75	ConnectionFactory SSL parameters	474
Table 76	Qualifiers for Cipher Suites in Java Clients	476
Table 77	OpenSSL Qualifiers for Cipher Suites	477
Table 78	Supported Cipher Suites in Java API.....	478
Table 79	Shared Storage Criteria for Fault Tolerance	495
Table 80	SSL Parameters for Routes	520
Table 81	Monitor topics.....	534
Table 82	Message properties	537
Table 83	Event Action Property Values	542
Table 84	Event Reason Property Values	544

Preface

TIBCO is proud to announce the latest release of TIBCO Enterprise Message Service™. This release is the latest in a long history of TIBCO products that leverage the power of the Information Bus® to enable truly event-driven IT environments. To find out more about how TIBCO Enterprise Message Service and other TIBCO products are powered by TIB® technology, please visit us at www.tibco.com.

TIBCO Enterprise Message Service software lets application programs send and receive messages according to the Java Message Service (JMS) protocol. It also integrates with TIBCO Rendezvous and TIBCO SmartSockets messaging products.

Topics

- [Changes from the Previous Release of this Guide, page xxvi](#)
- [Related Documentation, page xxviii](#)
- [Typographical Conventions, page xxx](#)
- [Connecting with TIBCO Resources, page xxxiii](#)

Changes from the Previous Release of this Guide

This section itemizes the major changes from the previous release of this guide.

Feature Enhancements

The following enhancements are documented in the sections shown:

- **Logging Enhancement** A new `tibemsd` parameter has been introduced that allows you to specify the maximum number of log files you want to keep. See the description for [logfile_max_count on page 221](#).

Support for JMS 2.0

This release adds support for the JMS 2.0 specification. Currently, this support is offered only to Java clients. The features added with JMS 2.0 include:

- **Delivery Delay** Message publishers can now specify a delivery time for messages. The EMS server will only deliver the message after the time delivery time specified when the message is published. For more information, see [Delivery Delay on page 91](#).
- **Asynchronous Sending** Message producers can now send messages asynchronously, offloading the notification of the success or failure to another thread and thereby increasing performance in certain situations. For details, see [Sending Messages Synchronously and Asynchronously on page 46](#).
- **Shared Subscriptions** An application can now share the work of message consumption across multiple topic consumers. When message consumers share a subscription to a topic, only one consumer will receive a published message. For details, see [Shared Subscriptions for Topics on page 5](#).
- **Simplified API** In addition to the API provided with the JMS 1.1 specification, which is now called the Classic API, the JMS 2.0 specification offers a simpler and less verbose API called the Simplified API. For details, see [JMS 2.0 Specification on page 318](#).

Changes in Functionality

- **Administration Tool Commands and Topic Consumers** With this release and the introduction of shared subscriptions, the relationship between topic subscriptions and topic consumers has changed. Most importantly, the number of subscriptions to a topic is not always equal to the number of consumers.

As a result, the output produced by some administration tool commands has changed:

- `show topics` — now reports the number of subscriptions and durable subscriptions, not the number of consumers.
- `show topic` — reports the number of subscriptions, durable subscriptions, and consumers. The number of consumers represents the number of *active* (that is non-closed) consumer objects created by applications. Offline or closed durable consumers are not included in the count.
- `show consumers` and `show stat [consumers]` — no longer report offline durable subscribers.

Refer to [Chapter 6, Using the EMS Administration Tool](#) for details on these commands.

Related Documentation

This section lists documentation resources you may find useful.

TIBCO Enterprise Message Service Documentation

The following documents form the TIBCO Enterprise Message Service documentation set:

- *TIBCO Enterprise Message Service User's Guide* Read this manual to gain an overall understanding of the product, its features, and configuration.
- *TIBCO Enterprise Message Service Central Administration* Read this manual for information on the central administration interface.
- *TIBCO Enterprise Message Service Installation* Read the relevant sections of this manual before installing this product.
- *TIBCO Enterprise Message Service C & COBOL Reference* The C API reference is available in HTML and PDF formats.
- *TIBCO Enterprise Message Service Java API Reference* The Java API reference can be accessed only through the HTML documentation interface.
- *TIBCO Enterprise Message Service .NET API Reference* The .NET API reference can be accessed only through the HTML documentation interface.
- *TIBCO Enterprise Message Service Release Notes* Read the release notes for a list of new and changed features. This document also contains lists of known issues and closed issues for this release. This document is available only in PDF format.

Other TIBCO Product Documentation

You may find it useful to read the documentation for the following TIBCO products:

- TIBCO Rendezvous[®]
- TIBCO SmartSockets[®]
- TIBCO Hawk[®]
- TIBCO EMS[®] Client for z/OS (CICS)
- TIBCO EMS[®] Client for z/OS (MVS)
- TIBCO EMS[®] Client for IBM i

Third Party Documentation

- Java™ Message Service specification, available through <http://www.oracle.com/technetwork/java/jms/index.html>.
- *Java™ Message Service* by Richard Monson-Haefel and David A. Chappell, O'Reilly and Associates, Sebastopol, California, 2001.
- Java™ Authentication and Authorization Service (JAAS) *LoginModule Developer's Guide* and *Reference Guide*, available through <http://www.oracle.com/technetwork/java/javase/jaas/index.html>.

Typographical Conventions

The following typographical conventions are used in this manual.

Table 1 General Typographical Conventions

Convention	Use
<i>TIBCO_HOME</i> <i>ENV_HOME</i> <i>EMS_HOME</i>	<p>Many TIBCO products must be installed within the same home directory. This directory is referenced in documentation as <i>TIBCO_HOME</i>. The value of <i>TIBCO_HOME</i> depends on the operating system. For example, on Windows systems, the default value is C:\tibco.</p> <p>Other TIBCO products are installed into an installation environment. Incompatible products and multiple instances of the same product are installed into different installation environments. The directory into which such products are installed is referenced in documentation as <i>ENV_HOME</i>. The value of <i>ENV_HOME</i> depends on the operating system. For example, on Windows systems the default value is C:\tibco.</p> <p>TIBCO Enterprise Message Service installs into a directory within <i>TIBCO_HOME</i>. This directory is referenced in documentation as <i>EMS_HOME</i>. The value of <i>EMS_HOME</i> depends on the operating system. For example on Windows systems, the default value is C:\tibco\ems\8.0.</p>
code font	<p>Code font identifies commands, code examples, filenames, pathnames, and output displayed in a command window. For example:</p> <p>Use MyCommand to start the foo process.</p>
bold code font	<p>Bold code font is used in the following ways:</p> <ul style="list-style-type: none">• In procedures, to indicate what a user types. For example: Type admin.• In large code samples, to indicate the parts of the sample that are of particular interest.• In command syntax, to indicate the default parameter for a command. For example, if no parameter is specified, MyCommand is enabled: MyCommand [enable disable]

Table 1 General Typographical Conventions (Cont'd)




Convention	Use
<i>italic font</i>	<p>Italic font is used in the following ways:</p> <ul style="list-style-type: none"> To indicate a document title. For example: See <i>TIBCO ActiveMatrix BusinessWorks Concepts</i>. To introduce new terms. For example: A portal page may contain several portlets. <i>Portlets</i> are mini-applications that run in a portal. To indicate a variable in a command or code syntax that you must replace. For example: <code>MyCommand PathName</code>
Key combinations	<p>Key name separated by a plus sign indicate keys pressed simultaneously. For example: Ctrl+C.</p> <p>Key names separated by a comma and space indicate keys pressed one after the other. For example: Esc, Ctrl+Q.</p>
	The note icon indicates information that is of special interest or importance, for example, an additional action required only in certain circumstances.
	The tip icon indicates an idea that could be useful, for example, a way to apply the information provided in the current section to achieve a specific result.
	The warning icon indicates the potential for a damaging situation, for example, data loss or corruption if certain steps are taken or not taken.

Table 2 Syntax Typographical Conventions

Convention	Use
[]	<p>An optional item in a command or code syntax.</p> <p>For example:</p> <p><code>MyCommand [optional_parameter] required_parameter</code></p>
	<p>A logical OR that separates multiple items of which only one may be chosen.</p> <p>For example, you can select only one of the following parameters:</p> <p><code>MyCommand para1 param2 param3</code></p>

Table 2 Syntax Typographical Conventions

Convention	Use
{ }	<p>A logical group of items in a command. Other syntax notations may appear within each logical group.</p> <p>For example, the following command requires two parameters, which can be either the pair param1 and param2, or the pair param3 and param4.</p> <pre>MyCommand {param1 param2} {param3 param4}</pre> <p>In the next example, the command requires two parameters. The first parameter can be either param1 or param2 and the second can be either param3 or param4:</p> <pre>MyCommand {param1 param2} {param3 param4}</pre> <p>In the next example, the command can accept either two or three parameters. The first parameter must be param1. You can optionally include param2 as the second parameter. And the last parameter is either param3 or param4.</p> <pre>MyCommand param1 [param2] {param3 param4}</pre>

Connecting with TIBCO Resources

How to Join TIBCOCommunity

TIBCOCommunity is an online destination for TIBCO customers, partners, and resident experts. It is a place to share and access the collective experience of the TIBCO community. TIBCOCommunity offers forums, blogs, and access to a variety of resources. To register, go to <http://www.tibcommunity.com>.

How to Access TIBCO Documentation

You can access TIBCO documentation here:

<http://docs.tibco.com>

How to Contact TIBCO Support

For comments or problems with this manual or the software it addresses, contact TIBCO Support as follows:

- For an overview of TIBCO Support, and information about getting started with TIBCO Support, visit this site:

<http://www.tibco.com/services/support>

- If you already have a valid maintenance or support contract, visit this site:

<https://support.tibco.com>

Entry to this site requires a user name and password. If you do not have a user name, you can request one.

Chapter 1 **Overview**

This chapter contains a general overview of Java Message Service (JMS) and TIBCO Enterprise Message Service concepts.

Topics

- [JMS Overview, page 2](#)
- [JMS Message Models, page 3](#)
- [EMS Destination Features, page 8](#)
- [Client APIs, page 10](#)
- [Administration, page 11](#)
- [Security, page 13](#)
- [Fault Tolerance, page 13](#)
- [Routing, page 14](#)
- [Integrating With Third-Party Products, page 14](#)

JMS Overview

Java Message Service (JMS) is a Java framework specification for messaging between applications. This specification was developed to supply a uniform messaging interface among enterprise applications.

Using a message service allows you to integrate the applications within an enterprise. For example, you may have several applications: one for customer relations, one for product inventory, and another for raw materials tracking. Each application is crucial to the operation of the enterprise, but even more crucial is communication between the applications to ensure the smooth flow of business processes. Message-oriented-middleware (MOM) creates a common communication protocol between these applications and allows you to easily integrate new and existing applications in your enterprise computing environment.

The JMS framework (an interface specification, not an implementation) is designed to supply a basis for MOM development. TIBCO Enterprise Message Service implements JMS and integrates support for connecting other message services, such as TIBCO Rendezvous and TIBCO SmartSockets. This chapter describes the concepts of JMS and its implementation in TIBCO Enterprise Message Service. For more information on JMS requirements and features, see the following sources:

- Java Message Service specification, available through <http://www.oracle.com/technetwork/java/jms/index.html>.
- *Java Message Service* by Richard Monson-Haefel and David A. Chappell, O'Reilly and Associates, Sebastopol, California, 2001.

JMS Compliance

TIBCO Enterprise Message Service 8.0 has passed Oracle Technology Compatibility Kit (TCK) for Java Message Service 2.0 (JMS 2.0). Therefore, EMS 8.0 is compliant with the JMS 2.0 specification, assuming the following requirements are met:

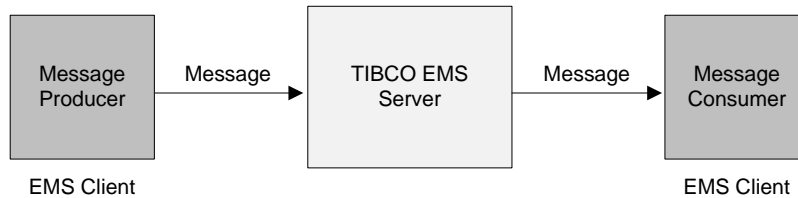
- Both the Java client and EMS server must be software release 8.0 or higher.
- All EMS software must be run on a supported operating system. Supported systems are listed in the readme file.
- The EMS software must be properly installed to include correct versions of software the EMS is dependent on.
- The EMS server configuration parameter `jms_2_0_compliance` must be set to `true`.

JMS Message Models

JMS is based on creation and delivery of messages. Messages are structured data that one application sends to another. The creator of the message is known as the *producer* and the receiver of the message is known as the *consumer*. The TIBCO EMS server acts as an intermediary for the message and manages its delivery to the correct destination. The server also provides enterprise-class functionality such as fault-tolerance, message routing, and communication with other messaging systems, such as TIBCO Rendezvous® and TIBCO SmartSockets®.

Figure 1 illustrates an application producing a message, sending it by way of the server, and a different application receiving the message.

Figure 1 Message Delivery



JMS supports these messaging models:

- [Point-to-Point](#) (queues)
- [Publish and Subscribe](#) (topics)
- [Multicast](#) (topics)

Point-to-Point

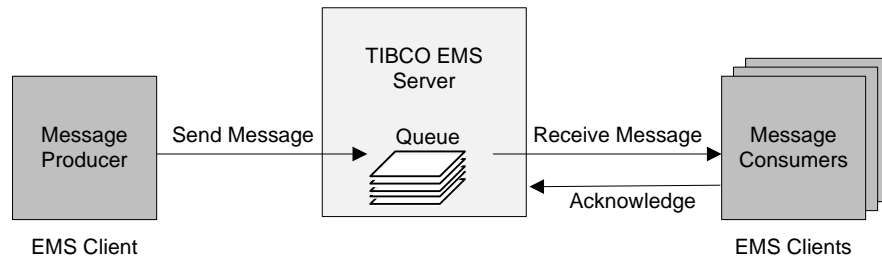
Point-to-point messaging has one producer and one consumer per message. This style of messaging uses a *queue* to store messages until they are received. The message producer sends the message to the queue; the message consumer retrieves messages from the queue and sends acknowledgement that the message was received.

More than one producer can send messages to the same queue, and more than one consumer can retrieve messages from the same queue. The queue can be configured to be exclusive, if desired. If the queue is exclusive, then all queue messages can only be retrieved by the first consumer specified for the queue. Exclusive queues are useful when you want only one application to receive messages for a specific queue. If the queue is not exclusive, any number of

receivers can retrieve messages from the queue. Non-exclusive queues are useful for balancing the load of incoming messages across multiple receivers. Regardless of whether the queue is exclusive or not, only one consumer can ever consume each message that is placed on the queue.

Figure 2 illustrates point-to-point messaging using a non-exclusive queue. Each message consumer receives a message from the queue and acknowledges receipt of the message. The message is taken off the queue so that no other consumer can receive it.

Figure 2 Point-to-point messages



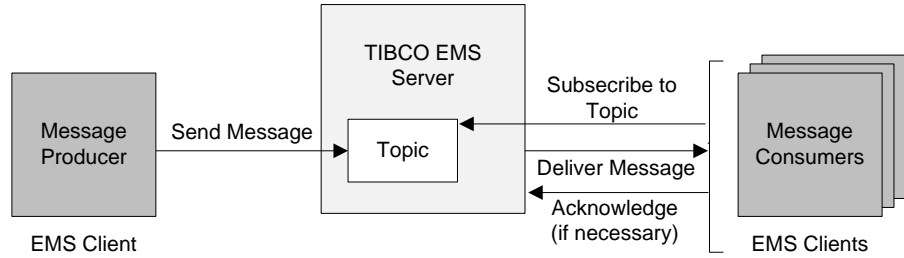
Publish and Subscribe

In a publish and subscribe message system, producers address messages to a *topic*. In this model, the producer is known as a *publisher* and the consumer is known as a *subscriber*.

Many publishers can publish to the same topic, and a message from a single publisher can be received by many subscribers. Subscribers subscribe to topics, and all messages published to the topic are received by all subscribers to the topic. This type of message protocol is also known as *broadcast* messaging because messages are sent over the network and received by all interested subscribers, similar to how radio or television signals are broadcast and received.

Figure 3 illustrates publish and subscribe messaging. Each message consumer subscribes to a topic. When a message is published to that topic, all subscribed consumers receive the message.

Figure 3 Publish and subscribe messages



Durable Subscribers for Topics

By default, subscribers only receive messages when they are active. If messages arrive on the topic when the subscriber is not available, the subscriber does not receive those messages.

The EMS APIs allow you to create durable subscribers to ensure that messages are received, even if the message consumer is not currently running. Messages for durable subscriptions are stored on the server as long as durable subscribers exist for the topic, or until the message expiration time for the message has been reached, or until the storage limit has been reached for the topic. Durable subscribers can receive messages from a durable subscription even if the subscriber was not available when the message was originally delivered.

When an application restarts and recreates a durable subscriber with the same ID, all messages stored on the server for that topic are delivered to the durable subscriber.

See [Creating a Message Consumer on page 346](#) for details on how to create durable subscribers.

Shared Subscriptions for Topics



This feature is currently available only with EMS clients for Java.

Shared subscriptions allow an application to share the work of receiving messages on a topic among multiple message consumers. When multiple consumers share a subscription, only one consumer in the group receives each new message. This is similar in function to a queue; however, there are no restrictions placed on the type of consumers to the topic, meaning that a topic can have a mix of shared and not shared, durable and non-durable consumers. When a message is published to the topic, the same message goes to all the matching subscriptions.

Shared subscriptions are created with a specific name, and optionally a client ID. Consumers sharing the subscription specify this name when subscribing to the topic. If the shared subscription type is durable, it persists and continues to accumulate messages until deleted. If the shared subscription type is non-durable, it persists only so long as subscribers exist.

For example, the topic `foo` might have the following subscriptions:

- not shared, non-durable subscription
- not shared, durable subscription
- shared, non-durable subscription called `mySharedSub` with three shared consumers
- shared, durable subscription called `myDurableSharedSub` with two shared consumers

If a message is received on `foo`, each of the above four subscriptions receive that same message. For the shared subscriptions `mySharedSub` and `myDurableSharedSub`, the message is delivered to only one of its respective shared consumers.

If the shared consumers of the shared durable subscription `myDurableSharedSub` are closed, then the shared durable subscription continues to exist and accumulate messages until it is deleted, or until the application creates a new durable shared consumer named `myDurableSharedSub` to resume this subscription. If the shared consumers of `mySharedSub` are all closed, the subscription is removed from topic `foo`.



Shared subscriptions cannot be used with multicast-enabled topics. That is, if a topic has a `channel` property set, shared subscriptions are not supported.

See [Creating a Message Consumer on page 346](#) for details on how to create shared subscriptions.

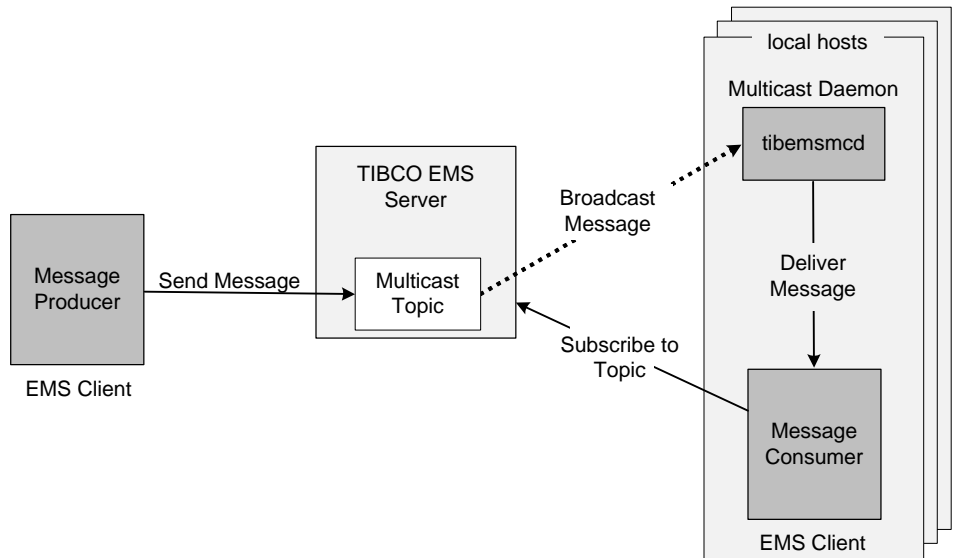
Multicast

Multicast messaging allows one message producer to send a message to multiple subscribed consumers simultaneously. As in the publish and subscribe messaging models, the message producer addresses the message to a topic. Instead of delivering a copy of the message to each individual subscriber over TCP, however, the EMS server broadcasts the message over Pragmatic General Multicast (PGM). A daemon running on the machine with the subscribed EMS client receives the multicast message and delivers it to the message consumer.

Multicast is highly scalable because of the reduction in bandwidth used to broadcast messages, and because of reduced EMS server resources used. However, multicast does not guarantee message delivery to all subscribers.

[Figure 4 on page 7](#) illustrates the multicast messaging model. Each message consumer subscribes to a multicast-enabled topic. When a message is sent to that topic, the EMS server broadcasts the message. Listening multicast daemons receive the message and deliver it to subscribed clients.

Figure 4 Multicast messages



For more information about multicast, see [Chapter 13, Using Multicast](#), on [page 369](#).

EMS Destination Features

TIBCO Enterprise Message Service allows you to configure destinations to enhance the functionality of each messaging model.

The EMS destination features allow you to:

- Set a [secure](#) mode for access control at the queue or topic level, so that some destinations may require permission and others may not. See [Destination Control on page 276](#).
- Set threshold limits for the amount of memory used by the EMS server to store messages for a topic or a queue and fine-tune the server's response to when the threshold is exceeded. See [flowControl on page 61](#) and [overflowPolicy on page 65](#).
- Route messages sent to destinations to other servers. See [Working With Routes on page 509](#).
- Create bridges between destinations of the same or different types to create a hybrid messaging model for your application. This can be useful if your application requires that you send the same message to both a topic and a queue. For more information on creating bridges between destinations and situations where this may be useful, see [Destination Bridges on page 82](#).
- Control the flow of messages to a destination. This is useful when message producers send messages much faster than message consumers can receive them. For more information on flow control, see [Flow Control on page 87](#).
- Exchange messages with other message services. Queues can receive TIBCO Rendezvous and TIBCO SmartSockets messages. Topics can either receive or send Rendezvous and TIBCO SmartSockets messages. See [Working With TIBCO Rendezvous on page 401](#) and [Working With TIBCO SmartSockets on page 425](#).
- Set queues to be exclusive or non-exclusive. Only one receiver can receive messages from an exclusive queue. More than one receiver can receive messages from non-exclusive queues. See [exclusive on page 59](#).
- Specify a redelivery policy for queues. When messages must be redelivered, you can specify a property on the queue that determines the maximum number of times a message should be redelivered. See [maxRedelivery on page 64](#).
- Trace and log all messages passing through a destination. See [trace on page 74](#).
- Include the user name of the message producer in the message. See [sender_name on page 72](#) and [sender_name_enforced on page 72](#).

- Administrator operations can use wildcards in destination names. The wildcard destination name is the parent, and any names that match the wildcard destination name inherit the properties of the parent. See [Wildcards on page 77](#).
- Use the `store` property to cause messages sent to a destination to be written to a store file. Set the destination store to `store=$sys.fail-safe` to direct the server to write messages to the file synchronously and guarantee that messages are not lost under any circumstances. See [store on page 73](#) for more information.
- Specify that a consumer is to receive batches of messages in the background to improve performance. Alternatively, you can specify that queue receivers are to only receive one message at a time. See [prefetch on page 68](#) for more information.

Client APIs

Java applications use the `javax.jms` package to send or receive JMS messages. This is a standard set of interfaces, specified by the JMS specification, for creating the connection to the EMS server, specifying the type of message to send, and creating the destination (topic or queue) on which to send or receive messages. You can find a description of the `javax.jms` package in *TIBCO Enterprise Message Service Java API Reference* included in the online documentation. Because EMS implements the JMS standard, you can also view the documentation on these interfaces along with the JMS specification at <http://www.oracle.com/technetwork/java/jms/index.html>.

TIBCO Enterprise Message Service includes parallel APIs for other development environments. See the following for more information:

- *TIBCO Enterprise Message Service C & COBOL API Reference*
- *TIBCO Enterprise Message Service .NET API Reference* (online documentation)

Sample Code

EMS includes several example programs. These examples illustrate various features of EMS. You may wish to view these example programs when reading about the corresponding features in this manual. The examples are included in the `samples` subdirectory of the EMS installation directory.

For more information about running the examples, see [Chapter 4, Getting Started, on page 93](#).

TIBCO Rendezvous Java Applications

EMS includes a Java class that allows pure Java TIBCO Rendezvous applications to connect directly with the EMS server; see [Pure Java Rendezvous Programs on page 423](#).

Administration

EMS provides mechanisms for administering server operations and creating objects that are managed by the server, such as ConnectionFactories and Destinations.

Administration functions can be issued either using the command-line administration tool or by creating an application that uses the administration API (either Java or .NET). The command-line administration tool is described in [Chapter 6, Using the EMS Administration Tool, on page 123](#). The administration APIs are described in the online documentation.

The administration interfaces allow you to create and manage administered objects such as ConnectionFactories, Topics, and Queues. EMS clients can retrieve references to these administered objects by using Java Naming and Directory Interface (JNDI). Creating static administered objects allows clients to use these objects without having to implement the objects within the client.

Administering the Server

EMS has several administration features that allow you to monitor and manage the server. The following table provides a summary of administration features and details where in the documentation you can find more information.

Table 3 Summary of administration features (Sheet 1 of 2)

Feature	More Information
Configuration files allow you to specify server characteristics.	Chapter 7, Using the Configuration Files, on page 185
Administration tool provides a command line interface for managing the server.	Chapter 6, Using the EMS Administration Tool, on page 123
Authentication and permissions can restrict access to the server and to destinations. You can also specify who can perform administrative activities with administrator permissions.	Chapter 8, Authentication and Permissions, on page 265
Configure log files to provide information about various server activity.	Chapter 17, Monitoring Server Activity, on page 445

Table 3 Summary of administration features (Sheet 2 of 2)

Feature	More Information
The server can publish messages when various system events occur. This allows you to create robust monitoring applications that subscribe to these system monitor topics.	Chapter 17, Monitoring Server Activity, on page 445
The server can provide various statistics at the desired level of detail.	Chapter 17, Monitoring Server Activity, on page 445

User and Group Management

EMS provides facilities for creating and managing users and groups locally for the server. The EMS server can also use an external system, such as an LDAP server for authenticating users and storing group information. See [Chapter 8, Authentication and Permissions, on page 265](#) for more information about configuring EMS to work with external systems for user and group management.

Using TIBCO Hawk

You can use TIBCO Hawk® for monitoring and managing the EMS server. See TIBCO Hawk documentation for more information.

Security

For communication security between servers and clients, and between servers and other servers, you must explicitly configure SSL within EMS.

Secure Sockets Layer (SSL) is a protocol for transmitting encrypted data over the Internet or an internal network. SSL works by using public and private keys to encrypt data that is transferred over the SSL connection. Most web browsers support SSL, and many Web sites and Java applications use the protocol to obtain confidential user information, such as credit card numbers.

EMS supports SSL between the following components:

- between an EMS client and the EMS server
- between the administration tool and the EMS server
- between the administration APIs and the EMS server
- between routed servers
- between fault-tolerant servers

See [Chapter 18, Using the SSL Protocol, on page 465](#) for more information about SSL support in EMS.

Fault Tolerance

You can configure EMS servers as primary and backup servers to provide fault tolerance for your environment. The primary and backup servers act as a pair, with the primary server accepting client connections and performing the work of handling messages, and the secondary server acting as a backup in case of failure. When the active server fails, the backup server assumes operation and becomes the primary active server.

See [Chapter 19, Fault Tolerance, on page 485](#) for more information about the fault-tolerance features of EMS.

Routing

EMS provides the ability for servers to route messages between each other. Topic messages can be routed across multiple hops, provided there are no cycles (that is, the message can not be routed to any server it has already visited). Queue messages can travel at most one hop to any other server from the server that owns the queue.

EMS stores and forwards messages in most situations to provide operation when a route is not connected.

See [Chapter 20, Working With Routes, on page 509](#) for more information about the routing features of EMS.

Integrating With Third-Party Products

EMS allows you to work with third-party naming/directory service products or with third-party application servers.

Transaction Support

TIBCO Enterprise Message Service can integrate with Java Transaction API (JTA) compliant transaction managers. EMS implements all interfaces necessary to be JTA compliant. The EMS C API is compliant with the X/Open XA specification. The EMS .NET API supports Microsoft Distributed Transaction Coordinator (MS DTC). Transactions created using MSDTC in a .NET client are seen as XA transactions in C and Java clients.

Chapter 2 **Messages**

This chapter provides an overview of EMS messages.

Topics

- [EMS Extensions to JMS Messages, page 16](#)
- [JMS Message Structure, page 17](#)
- [Message Priority, page 24](#)
- [Message Delivery Modes, page 25](#)
- [How EMS Manages Persistent Messages, page 27](#)
- [Store Messages in Multiple Stores, page 30](#)
- [Character Encoding in Messages, page 36](#)
- [Message Compression, page 38](#)
- [Message Acknowledgement, page 39](#)
- [Sending Messages Synchronously and Asynchronously, page 46](#)

EMS Extensions to JMS Messages

The JMS specification details a standard format for the header and body of a message. Properties are provider-specific and can include information on specific implementations or enhancements to JMS functionality. See [EMS Message Properties on page 19](#) for the list of message properties that are specific to EMS.

In addition to the EMS message properties, EMS provides a select number of extensions to JMS. These are:

- The JMS standard specifies two delivery modes for messages, `PERSISTENT` and `NON_PERSISTENT`. EMS also includes a `RELIABLE_DELIVERY` mode that eliminates some of the overhead associated with the other delivery modes. See [RELIABLE_DELIVERY on page 26](#).
- For consumer sessions, you can specify a `NO_ACKNOWLEDGE` mode so that consumers do not need to acknowledge receipt of messages, if desired. EMS also provides an `EXPLICIT_CLIENT_ACKNOWLEDGE` and `EXPLICIT_CLIENT_DUPS_OK_ACKNOWLEDGE` mode that restricts the acknowledgement to single messages. See [Message Acknowledgement on page 39](#).
- EMS extends the [MapMessage](#) and [StreamMessage](#) body types. These extensions allow EMS to exchange messages with TIBCO Rendezvous and ActiveEnterprise formats that have certain features not available within the JMS `MapMessage` and `StreamMessage`.

TIBCO Enterprise Message Service adds these two extensions to the `MapMessage` and `StreamMessage` body types:

- You can insert another `MapMessage` or `StreamMessage` instance as a submessage into a `MapMessage` or `StreamMessage`, generating a series of nested messages, instead of a flat message.
- You can use arrays as well as primitive types for the values.

These extensions add considerable flexibility to the `MapMessage` and `StreamMessage` body types. However, they are extensions and therefore not compliant with JMS specifications. Extended messages are tagged as extensions with the vendor property tag `JMS_TIBCO_MSG_EXT`.

For more information on compatibility with Rendezvous messages, see [Message Body on page 419](#).

JMS Message Structure

JMS messages have a standard structure. This structure includes the following sections:

- Header (required)
- Properties (optional)
- Body (optional)

JMS Message Header Fields

The header contains 11 predefined fields that contain values used to route and deliver messages. [Table 4](#) describes the message header fields.

Table 4 JMS Message Headers

Header Field	Set by	Comments
JMSDestination	send or publish method	Destination to which message is sent
JMSDeliveryMode	send or publish method	Persistent or non-persistent message. The default is persistent. EMS extends the delivery mode to include a RELIABLE_DELIVERY mode, as described in RELIABLE_DELIVERY on page 26 .

Table 4 JMS Message Headers

Header Field	Set by	Comments
JMSExpiration	send or publish method	<p>Length of time that message will live before expiration. If set to 0, message does not expire. The time-to-live is specified in milliseconds.</p> <p>If the server expiration property is set for a destination, it will override the JMSExpiration value set by the message producer.</p> <p>In EMS version 4.4 and later, clients automatically synchronize their clocks with the server when a connection is created. However, for long-lasting connections, Network Time Protocol (NTP) is the most reliable method for ensuring continuing synchronization between server and client. Additionally, if your EMS server or client application are based on a version of EMS prior to 4.4, you must ensure that clocks are synchronized among all the host computers that send and receive messages, if your client application uses non-zero values for message expiration. Synchronize clocks to a tolerance that is a very small fraction of the smallest message expiration time.</p>
JMSDeliveryTime	send or publish method	<p>Read-only field. If the message producer has a delivery delay set, then the time returned here after calling the send method represents the earliest time when the EMS server will deliver the message to consumers. Once the message has been received, it carries that same value. This value is calculated by adding the delivery delay value held by the message producer to the time the message was sent. For transactions, the delivery time is calculated using the time the client sends the message, not the time the transaction is committed.</p> <p>For more information, see Delivery Delay on page 91.</p>
JMSPriority	send or publish method	<p>Uses a numerical ranking, between 0 and 9, to define message priority as normal or expedited. Larger numbers represent higher priority.</p> <p>See Message Priority on page 24 for more information.</p>
JMSMessageID	send or publish method	<p>Value uniquely identifies each message sent by a provider.</p>

Table 4 JMS Message Headers

Header Field	Set by	Comments
JMSTimestamp	send or publish method	Timestamp of time when message was handed off to a provider to be sent. Message may actually be sent later than this timestamp.
JMSCorrelationID	message client	This ID can be used to link messages, such as linking a response message to a request message. Entering a value in this field is optional. The JMS Correlation ID has a recommended maximum of 4 KB. Higher values may result in the message being rejected.
JMSReplyTo	message client	A destination to which a message reply should be sent. Entering a value for this field is optional.
JMSType	message client	Message type identifier.
JMSRedelivered	JMS provider	If this field is set, it is possible that the message was delivered to the client earlier, but not acknowledged at that time.

EMS Message Properties

In the properties area, applications, vendors, and administrators on JMS systems can add optional properties. The properties area is optional, and can be left empty. The JMS specification describes the JMS message properties. This section describes the message properties that are specific to EMS.

TIBCO-specific property names begin with JMS_TIBCO. Client programs may use the TIBCO-specific properties to access EMS features, but not for communicating application-specific information among client programs.

The EMS properties are summarized in [Table 5](#) and described in more detail in subsequent sections in this chapter.

Table 5 Summary of message properties (Sheet 1 of 2)

Property	Description	More Info
JMS_TIBCO_CM_PUBLISHER	Correspondent name of an RVCN sender for messages imported from TIBCO Rendezvous.	418

Table 5 Summary of message properties (Sheet 2 of 2)

Property	Description	More Info
JMS_TIBCO_CM_SEQUENCE	Sequence number of an RVCM message imported from TIBCO Rendezvous.	418
JMS_TIBCO_COMPRESS	Allows messages to be compressed for more efficient storage.	38
JMS_TIBCO_DISABLE_SENDER	Specifies that the user name of the message sender should not be included in the message, if possible.	21
JMS_TIBCO_IMPORTED	Set by the server when the message has been imported from Rendezvous or SmartSockets.	418 438
JMS_TIBCO_MSG_EXT	Extends the functionality of the MapMessage and StreamMessage body types to include submessages or arrays.	16 418 438
JMS_TIBCO_MSG_TRACE	Specifies the message should be traced from producer to consumer.	452
JMS_TIBCO_PRESERVE_UNDELIVERED	Specifies the message is to be placed on the undelivered message queue if the message must be removed.	21
JMS_TIBCO_SENDER	Contains the user name of the message sender.	21
JMS_TIBCO_SS_SENDER	When the EMS server imports a message from TIBCO SmartSockets, it sets this property to the SmartSockets sender header field (in SmartSockets syntax).	438

Undelivered Message Queue

If a message expires or has exceeded the value specified by the [maxRedelivery](#) property on a queue, the server checks the message's [JMS_TIBCO_PRESERVE_UNDELIVERED](#) property. If `JMS_TIBCO_PRESERVE_UNDELIVERED` is set to `true`, the server moves the message to the undelivered message queue, `$sys.undelivered`. This undelivered message queue is a system queue that is always present and cannot be deleted. If `JMS_TIBCO_PRESERVE_UNDELIVERED` is set to `false`, the message will be deleted by the server.

To make use of the undelivered message queue, the application that sends or publishes the message must set the boolean `JMS_TIBCO_PRESERVE_UNDELIVERED` property to `true` before sending or publishing the message.

You can only set the undelivered property on individual messages, there is no way to set the undelivered message queue as an option at the per-topic or per-queue level.

You should create a queue receiver to receive and handle messages as they arrive on the undelivered message queue. If you wish to remove messages from the undelivered message queue without receiving them, you can purge the `$sys.undelivered` queue with the administration tool, using the `purge queue` command described under [Command Listing on page 128](#). You can also remove messages using the administrative API included with TIBCO Enterprise Message Service.

Note that `$sys.undelivered` ignores the [global](#) destination property setting. Messages in the undelivered message queue are not routed to other servers.

Including the Message Sender

Within a message, EMS can supply the user name given by the message producer when a connection is created. The `sender_name` and `sender_name_enforced` server properties on the destination determine whether the message producer's user name is included in the sent message.

When a user name is included in a message, a message consumer can retrieve that user name by getting the string message property named [JMS_TIBCO_SENDER](#).

When the `sender_name` property is enabled and the `sender_name_enforced` property is not enabled on a destination, message producers can specify that the user name is to be left out of the message. Message producers can specify the [JMS_TIBCO_DISABLE_SENDER](#) boolean property for a particular message, and the message producer's user name will not be included in the message. However, if the `sender_name_enforced` property is enabled, the `JMS_TIBCO_DISABLE_SENDER` property is ignored and the user name is always included in the message.

JMS Message Bodies

A JMS message has one of several types of message bodies, or no message body at all.

The types of messages are described in [Table 6](#).

Table 6 JMS Message Types

Message Type	Contents of Message Body
Message	This message type has no body. This is useful for simple event notification.
TextMessage	A <code>java.lang.String</code> .
MapMessage	<p>A set of name/value pairs. The names are <code>java.lang.String</code> objects, and the values are Java primitive value types or their wrappers. The entries can be accessed sequentially by enumeration or directly by name. The order of entries is undefined.</p> <p>When EMS is exchanging messages with Rendezvous or ActiveEnterprise, you can generate a series of nested MapMessages, as described in EMS Extensions to JMS Messages on page 16.</p>
BytesMessage	A stream of uninterrupted bytes. The bytes are not typed; that is, they are not assigned to a primitive data type.
StreamMessage	<p>A stream of primitive values in the Java programming language. Each set of values belongs to a primitive data type, and must be read sequentially.</p> <p>When EMS is exchanging messages with Rendezvous or ActiveEnterprise, you can generate a series of nested StreamMessages, as described in EMS Extensions to JMS Messages on page 16.</p>
ObjectMessage	A serializable object constructed in the Java programming language.

Maximum Message Size

EMS supports messages up to a maximum size of 512MB. However, we recommend that application programs use smaller messages, since messages approaching this maximum size will strain the performance limits of most current hardware and operating system platforms.

Message Priority

The JMS specification includes a [JMSPriority](#) message header field in which senders can set the priority of a message, as a value in the range [0,9]. EMS *does* support message priority (though it is optional, and other vendors might not implement it).

When the EMS server has several messages ready to deliver to a consumer client, and must select among them, then it delivers messages with higher priority before those with lower priority.

However, priority ordering applies only when the server has a *backlog* of deliverable messages for a consumer. In contrast, when the server has only one message at a time to deliver to a consumer, then the priority ordering feature will not be apparent.

You can set default message priority for the Message Producer, as described in [Configuring a Message Producer on page 343](#). The default priority can be overridden by the client when sending a message, as described in [Sending Messages on page 354](#).

See Also JMS Specification, chapter 3.4.10

Message Delivery Modes

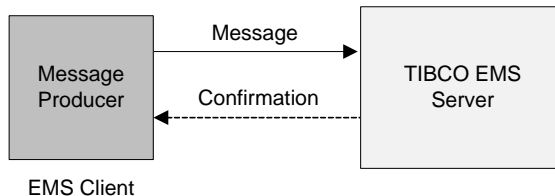
The `JMSDeliveryMode` message header field defines the delivery mode for the message. JMS supports `PERSISTENT` and `NON_PERSISTENT` delivery modes for both topic and queue. EMS extends these delivery modes to include a `RELIABLE_DELIVERY` mode.

You can set the default delivery mode for the Message Producer, as described in [Configuring a Message Producer on page 343](#). This default delivery mode can be overridden by the client when sending a message, as described in [Sending Messages on page 354](#).

PERSISTENT

As shown in [Figure 5](#), when a producer sends a `PERSISTENT` message, the producer must wait for the server to reply with a confirmation. The message is persisted on disk by the server. This delivery mode ensures delivery of messages to the destination on the server in almost all circumstances. However, the cost is that this delivery mode incurs two-way network traffic for each message or committed transaction of a group of messages.

Figure 5 Persistent Message Delivery



NON_PERSISTENT

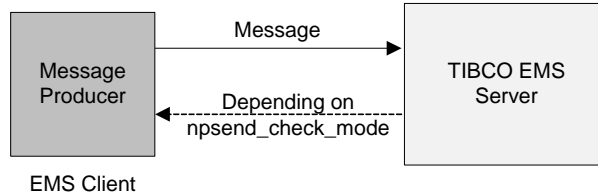
Sending a `NON_PERSISTENT` message omits the overhead of persisting the message on disk to improve performance.

If [authorization](#) is disabled on the server, the server does not send a confirmation to the message producer.

If [authorization](#) is enabled on the server, the default condition is for the producer to wait for the server to reply with a confirmation in the same manner as when using `PERSISTENT` mode.

Regardless of whether authorization is enabled or disabled, you can use the `npsend_check_mode` parameter in the `tibemsd.conf` file to specify the conditions under which the server is to send confirmation of `NON_PERSISTENT` messages to the producer. See the description for `npsend_check_mode` on page 202 for details.

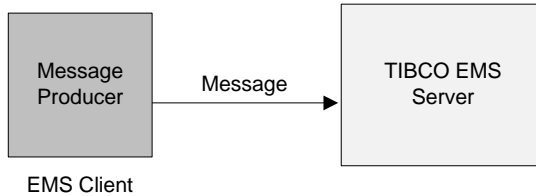
Figure 6 Non-Persistent Message Delivery



RELIABLE_DELIVERY

EMS extends the JMS delivery modes to include reliable delivery. Sending a `RELIABLE_DELIVERY` message omits the server confirmation to improve performance regardless of the `authorization` setting.

Figure 7 Reliable Message Delivery



When using `RELIABLE_DELIVERY` mode, the server never sends the producer a receipt confirmation or access denial and the producer does not wait for it. Reliable mode decreases the volume of message traffic, allowing higher message rates, which is useful for messages containing time-dependent data, such as stock price quotations.

When you use the reliable delivery mode, the client application does not receive any response from the server. Therefore, all publish calls will always succeed (not throw an exception) unless the connection to the server has been terminated.

In some cases a message published in reliable mode may be disqualified and not handled by the server because the destination is not valid or access has been denied. In this case, the message is not sent to any message consumer. However, unless the connection to the server has been terminated, the publishing application will not receive any exceptions, despite the fact that no consumer received the message.

How EMS Manages Persistent Messages

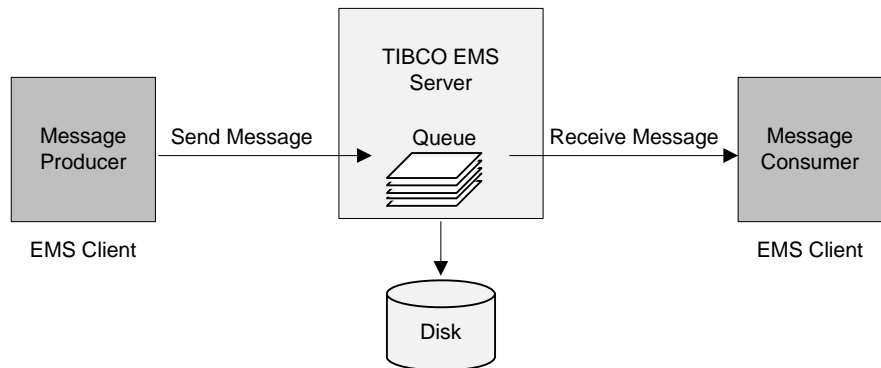
As described in [Message Delivery Modes on page 25](#), JMS defines two message delivery modes, `PERSISTENT` and `NON_PERSISTENT`, and EMS defines a `RELIABLE_DELIVERY` mode.

`NON_PERSISTENT` and `RELIABLE_DELIVERY` messages are never written to persistent storage. `PERSISTENT` messages are written to persistent storage when they are received by the EMS server.

Persistent Messages Sent to Queues

Persistent messages sent to a queue are always written to disk. Should the server fail before sending persistent messages to subscribers, the server can be restarted and the persistent messages will be sent to the subscribers when they reconnect to the server.

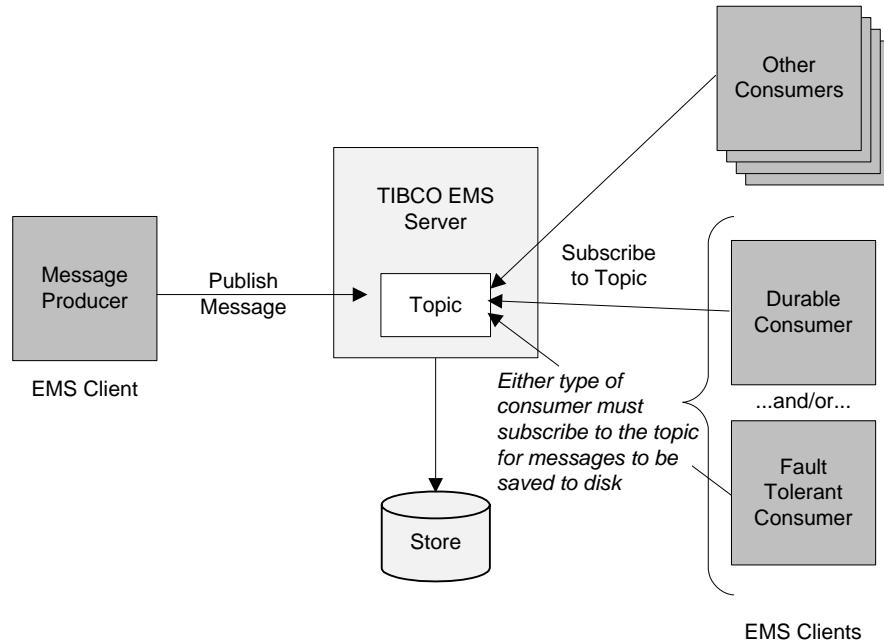
Figure 8 *Persistent Messages Sent to a Queue*



Persistent Messages Published to Topics

Persistent messages published to a topic are written to disk *only* if that topic has at least one durable subscriber or one subscriber with a fault-tolerant connection to the EMS server. In the absence of a durable subscriber or subscriber with a fault-tolerant connection, there are no subscribers that need messages resent in the event of a server failure. In this case, the server does not needlessly save persistent messages. This improves performance by eliminating the unnecessary disk I/O to persist the messages.

Figure 9 Persistent Messages Published to a Topic



This behavior is consistent with the JMS specification because durable subscribers to a topic cause published messages to be saved. Additionally, subscribers to a topic that have a fault-tolerant connection need to receive messages from the secondary server after a failover. However, non-durable subscribers without a fault-tolerant connection that re-connect after a server failure are considered newly created subscribers and are not entitled to receive any messages created prior to the time they are created (that is, messages published before the subscriber re-connects are not resent).

Persistent Messages and Synchronous File Storage

When using file storage, persistent messages received by the EMS server are by default written asynchronously to disk. This means that, when a producer sends a persistent message, the server does not wait for the write-to-disk operation to complete before returning control to the producer. Should the server fail before completing the write-to-disk operation, the producer has no way of detecting the failure to persist the message and taking corrective action.

You can set the `mode` parameter to `sync` for a given file storage in the [stores.conf](#) file to specify that persistent messages for the topic or queue be synchronously written to disk. When `mode = sync`, the persistent producer remains blocked until the server has completed the write-to-disk operation.

Each EMS server writes persistent messages to a store file. To prevent two servers from using the same store file, each server restricts access to its store file for the duration of the server process. For details on how EMS manages shared store files, see [How EMS Manages Access to Shared Store Files on page 120](#).

Store Messages in Multiple Stores

As described in [Message Delivery Modes on page 25](#), the EMS server writes PERSISTENT messages to disk while waiting for confirmation of receipt from the subscriber. Messages are persisted to a *store*. The EMS server can write messages to different types of stores: file-based stores, mstores, and database stores.

By default, the EMS server writes persistent messages to file-based stores. There are three default store files, as described in [Default Store Files on page 31](#). You can configure the system to change the default store files and locations, and also to store persistent messages to one or more store files, filtering them by destination. Stores are defined in the `stores.conf` configuration file, and associated with a destination using the `store` destination property.

Stores have properties that allow you to control how the server manages the store files. For example:

- When using file-based stores:
 - Preallocate disk space for the store file.
 - Truncate the file periodically to relinquish disk space.
 - Specify whether messages are written synchronously or asynchronously.
- Store messages in a database.

With the multiple stores feature, you can configure your messaging application to store messages in different locations for each application, or to create separate files for related destinations. For example, you can create one store for messages supporting Marketing, and one for messages supporting Sales. Because stores are configured in the server, they are transparent to clients.

The EMS Administration Tool allows administrators to review the system's configured stores and their settings by using the `show stores` and `show store` commands.

Store Types

TIBCO Enterprise Message Service allows you to configure several different types of stores, described here.

File-Based Stores

The EMS server stores persistent messages in file-based stores. You can use the default store files, or create your own file-based stores. You direct the EMS server to write messages to these store files by associating a destination with a store.

File-based stores are enabled by default, and the server automatically defines three default stores, described below. You do not need to do anything in order to use the default stores.

The section [Configuring Multiple Stores on page 32](#) describes how to change store settings or create custom stores.

mstores

The mstore is designed to recover quickly after a failover. When mstores are in use, the EMS server starts quickly, but may run more slowly until the mstore cache is fully loaded. This is because the EMS server continually monitors the store in the background. The server reads through the mstore incrementally and discards stale data, such as purged and expired messages.

As a result, expired and purged messages are not immediately removed from the mstore, and may remain in the store longer than they would in a file-based or database store—although they are not delivered to the consumer. These messages are discarded during the periodic scans of the store. The scanning behavior is determined by parameter settings in the store configuration, and is further described in [Understanding mstore Intervals on page 33](#).

Because of this behavior, querying the server for a total pending message count may return an inaccurate value. However, querying specific destinations returns an accurate count.

The section [Configuring Multiple Stores on page 32](#) describes the mstore configuration process.

Database Stores

The EMS server can store messages in one or more database instances. Database stores must be configured to use a supported database. See [Chapter 10, Using Database Stores](#) for a full description of this feature.

Default Store Files

The EMS server defines these default store files, and writes persistent messages and meta data to them:

- `$sys.nonfailsafe`—Persistent messages without a store property designation are written to `$sys.nonfailsafe` by default. The server writes messages to this store using asynchronous I/O calls.
- `$sys.failsafe`—Associate a destination with this store to write messages synchronously. The server writes messages to this store using synchronous I/O calls.

- `$sys.meta`—The server writes state information about durable subscribers, fault-tolerant connections, and other metadata in this store.

The EMS server creates these file-based stores automatically, and no steps are required to enable or deploy them. However, you can change the system configuration to customize the default store file settings, or even override the default store settings to either point to different file location, or write to an `mstore` or database.



Note that the `$sys.meta` store may not be reconfigured to use the `mstore` type.

Configuring Multiple Stores

This section describes the basic steps required to configure file-based stores and `mstores`. Database store configuration is detailed in [Chapter 10, Using Database Stores](#).

Settings for creating and configuring multiple stores are managed in the EMS server, and are transparent to clients. To configure the multiple stores feature, follow these steps:

1. Setup and configure stores in the `stores.conf` file.

Stores are created and configured in the `stores.conf` file. Each store must have a unique name. The stores are configured through parameters.

- File-based stores have two required parameters, `type` and `file`, which determine that the store is a file-based store, and set its location and filename. Optional parameters allow you to determine other settings, including how messages are written to the file, the minimum size of the file, and whether the EMS server attempts to truncate the file.
- `mstores` also have two required parameters, `type` and `file`. Optional parameters configure the scan interval, during which expired and purged messages are removed. See [Understanding mstore Intervals on page 33](#) below for information about interval settings.

2. Associate destinations with the configured stores.

Messages are sent to different stores according to their destinations. Destinations are associated with specific stores with the `store` parameter in the `topics.conf` and `queues.conf` files.

File-Based Stores

When using file-based stores, you can also change store associations dynamically using the `setprop topic` or `setprop queue` command in the EMS Administration Tool.

- mstores** When using mstores, you cannot dynamically change the mstore associations after they have been set. In order to change a destination's store property from a store of the type mstore:
- Stop the EMS server.
 - Empty the associated mstore of messages from the destination.
 - Change the store association by manually editing the destination's store property in the `topics.conf` or `queues.conf` file.
 - Restart the EMS server.



Once mstores are enabled for a destination, you cannot dynamically change the store property value using `setprop` or `addprop`. To change the store property, you must stop the server, empty the mstore, manually make the change, and restart.



The mstore stores data in multiple files. As a result, mstores cannot operate in out of space conditions. In order to prevent an out of space situation from arising, we recommend ensuring that there is at least twice as much disk space available for the mstore as needed to hold the maximum amount of data that might be stored in it.

Multiple destinations can be mapped to the same store, either explicitly or using wildcards. Even if no stores are configured, the server sends persistent messages that are not associated with a store to default stores. See [Default Store Files](#) for more information.

For details about the store parameter, see [store on page 73](#).

Understanding mstore Intervals

The mstore is designed to ensuring a quick EMS server start-up time. To enable this functionality, the EMS server must continually monitor the store in the background. The server reads through the mstore incrementally and discards stale data, such as purged and expired messages.

In order to keep the background activity from degrading server performance, the examination is performed in increments. The length of these increments and the amount of data processed each increment are controlled by two parameter settings. These `stores.conf` parameters can be configured for each mstore.

The default parameter settings are optimized for best performance in most production environments (see [mstore Parameters on page 256](#) for information about the default values). However, if the amount of data in the mstore grows significantly, the read rates associated with the background activity may begin to

affect message transmission rates in the EMS server. If the EMS server performance is negatively affected by the size of the mstore, you can tune the mstore parameter values to spread mstore background activity over a longer period of time, thereby decreasing the associated read rates.

- `scan_target_interval` — the maximum amount of time allowed before each message in the store is examined.

For example, if the `scan_target_interval` is 24 hours, each section of the mstore will be examined at least once every day. Because purged and expired messages are not removed from the mstore until they are examined by this background process, this means that it can take up to 24 hours before a message is removed from the queue following a purge command (making underlying storage space available for re-use).

- `scan_iter_interval` — the length of time between each increment of background activity.

For example, if the `scan_iter_interval` is 10 seconds, the EMS server begins examining a new section of the mstore every 10 seconds. The amount of data read in each increment is dependent on the total size of the store and the length of the `scan_target_interval`. The server must examine enough data in each interval to fully traverse the store within the target interval.

Example

For example, assume that `scan_iter_interval` is 10 seconds, `scan_target_interval` is 1 day (86,400 seconds), and the mstore contains 9 GBs of data. Every 10 seconds, the EMS server will examine about 1 MB of data. This produces an average read rate of about 100 KB/sec, which is unlikely to produce performance degradation with most modern storage mediums.

If EMS server performance does slow, you may need to increase the `scan_target_interval` value in order to spread the background activity over longer period of time. You can monitor the settings for problems using the `show store` command and checking the ratio of "Discard scan bytes" to "Discard scan interval". For best results, this ratio should be kept below 20% of the disk processing capacity for each mstore. Consider this ratio in relation to your storage medium's overall data transfer capacity, so as to make sure that the background activity does not occupy an excessive amount of the system's resources.

Implications for Statistics

The background monitoring and cleanup that occurs in the mstore also affect some key server statistics. Before the first scan has been completed, some message statistics may be underreported due to purged and expired messages that the server has not yet removed. Until the first background scan is complete for some or all mstores, the server may not have an accurate messages count.

For example, when the EMS server first starts, the "Pending Messages" and "Pending Message Size" counts reported by the `info` command in the administration tool can be understated, because the command only reports on messages it has scanned before the command is issued. Similarly, the "Message Count" and "Message Size" reported by the `show store` command may report a smaller number than actually exist in the store.

Once the first scan is complete, these counts can be considered accurate. To check the scan status on an mstore, use the `show store` command. The statistics returned now include a "First scan finished" field, which reports the scan status since the last EMS server start time. When the value of this field is `true`, the server statistics can be considered accurate.

If it is important to acquire the correct values for these statistics sooner, you will need to decrease the `scan_target_interval`.

Character Encoding in Messages

Character encodings are named sets of numeric values for representing characters. For example, ISO 8859-1, also known as Latin-1, is the character encoding containing the letters and symbols used by most Western European languages. If your applications are sending and receiving messages that use only English language characters (that is, the ASCII character set), you do not need to alter your programs to handle different character encodings. The EMS server and application APIs automatically handle ASCII characters in messages.

Character sets become important when your application is handling messages that use non-ASCII characters (such as the Japanese language). Also, clients encode messages by default as UTF-8. Some character encodings use only one byte to represent each character, but UTF-8 can potentially use two bytes to represent the same character. For example, the Latin-1 is a single-byte character encoding. If all strings in your messages contain only characters that appear in the Latin-1 encoding, you can potentially improve performance by specifying Latin-1 as the encoding for strings in the message.

EMS clients can specify a variety of common character encodings for strings in messages. The character encoding for a message applies to strings that appear in any of the following places within a message:

- property names and property values
- `MapMessage` field names and values
- data within the message body

The EMS client APIs (Java, .NET and C) include mechanisms for handling strings and specifying the character encoding used for all strings within a message. The following sections describe the implications of string character encoding for EMS clients.



Nearly all character sets include unprintable characters. EMS software does not prevent programs from using unprintable characters. However, messages containing unprintable characters (whether in headers or data) can cause unpredictable results if you instruct EMS to print them. For example, if you enable the message tracing feature, EMS prints messages to a trace log file.

Supported Character Encodings

Each message contains the name of the character encoding used to encode strings within the message. This character encoding name is one of the canonical names for character encodings contained in the Java specification. You can obtain a list of canonical character encoding names from the java.sun.com website.

Java and .NET clients use these canonical character encoding names when setting or retrieving the character encoding names. C clients have a list of macros that correspond to these canonical names. See the C API references for a list of supported character encodings in these interfaces.

Sending Messages

When a client sends a message, the message stores the character encoding name used for strings in that message. Java clients represent strings using Unicode. A message created by a Java client that does not specify an encoding will use UTF-8 as the named encoding within the message. UTF-8 uses up to four bytes to represent each character, so a Java client can improve performance by explicitly using a single-byte character encoding, if possible.

Java clients can globally set the encoding to use with the `setEncoding` method or the client can set the encoding for each message with the `setMessageEncoding` method. For more information about these methods, see the *TIBCO Enterprise Message Service Java API Reference*.

Typically, C clients manipulate strings using the character encoding of the machine on which they are running.

Message Compression

TIBCO Enterprise Message Service allows a client to compress the body of a message before sending the message to the server. EMS supports message compression/decompression across client types (Java, C and C#). For example, a Java producer may compress a message and a C consumer may decompress it.



Message compression is supported in .NET clients when using the install package for Visual C++ 8 / .NET 2.0. .NET in the Visual C++ 7 / .NET 1.1 package does not support compression.

About Message Compression

Message compression is especially useful when messages will be stored on the server (persistent queue messages, or topics with durable subscribers). Setting compression ensures that messages will take less memory space in storage. When messages are compressed and then stored, they are handled by the server in the compressed form. Compression assures that the messages will usually consume less space on disk and will be handled faster by the EMS server.

The compression option only compresses the body of a message. Headers and properties are never compressed. It is best to use compression when the message bodies will be large and the messages will be stored on a server.

When messages will not be stored, compression is not as useful. Compression normally takes time, and therefore the time to send or publish and receive compressed messages is generally longer than the time to send the same messages uncompressed. There is little purpose to message compression for small messages that are not be stored by the server.

Setting Message Compression

Message compression is specified for individual messages. That is, message compression, if desired, is set at the message level. TIBCO Enterprise Message Service does not define a way to set message compression at the per-topic or per-queue level.

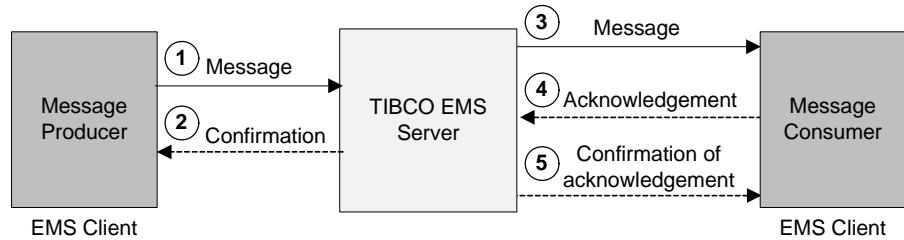
To set message compression, the application that sends or publishes the message must access the message properties and set the boolean property `JMS_TIBCO_COMPRESS` to `true` before sending or publishing the message.

Compressed messages are handled transparently. The client code only sets the `JMS_TIBCO_COMPRESS` property. The client does not need to take any other action. The client automatically decompresses any compressed messages it receives.

Message Acknowledgement

The interface specification for JMS requires that message delivery be guaranteed under many, but not all, circumstances. [Figure 10](#) illustrates the basic structure of message delivery and acknowledgement.

Figure 10 Message Delivery and Acknowledgement



The following describes the steps in message delivery and acknowledgement:

1. A message is sent from the message producer to the machine on which the EMS server resides.
2. For persistent messages, the EMS server sends a confirmation to the producer that the message was received.
3. The server sends the message to the consumer.
4. The consumer sends an acknowledgement to the server that the message was received. A session can be configured with a specific session mode that specifies how the consumer-to-server acknowledgement is handled. These session modes are described below.
5. In many cases, the server then sends a confirmation of the acknowledgement to the consumer.

The JMS specification defines three levels of acknowledgement for non-transacted sessions:

- `CLIENT_ACKNOWLEDGE` specifies that the consumer is to acknowledge all messages that have been delivered so far by the session. When using this mode, it is possible for a consumer to fall behind in its message processing and build up a large number of unacknowledged messages.
- `AUTO_ACKNOWLEDGE` specifies that the session is to automatically acknowledge consumer receipt of messages when message processing has finished.
- `DUPS_OK_ACKNOWLEDGE` specifies that the session is to "lazily" acknowledge the delivery of messages to the consumer. "Lazy" means that the consumer can delay acknowledgement of messages to the server until a convenient time;

meanwhile the server might redeliver messages. This mode reduces session overhead. Should JMS fail, the consumer may receive duplicate messages.

EMS extends the JMS session modes to include:

- `NO_ACKNOWLEDGE`
- `EXPLICIT_CLIENT_ACKNOWLEDGE`
- `EXPLICIT_CLIENT_DUPS_OK_ACKNOWLEDGE`



The Simplified JMS API introduced in JMS 2.0 supports the session modes defined in the JMS specification: `CLIENT_ACKNOWLEDGE`, `AUTO_ACKNOWLEDGE`, `DUPS_OK_ACKNOWLEDGE` and `SESSION_TRANSACTED`. However, it does not support the EMS extended session modes.

The session mode is set when creating a Session, as described in [Creating a Session on page 337](#).

NO_ACKNOWLEDGE

`NO_ACKNOWLEDGE` mode suppresses the acknowledgement of received messages. After the server sends a message to the client, all information regarding that message for that consumer is eliminated from the server. Therefore, there is no need for the client application to send an acknowledgement to the server about the received message. Not sending acknowledgements decreases the message traffic and saves time for the receiver, therefore allowing better utilization of system resources.



Sessions created in no-acknowledge receipt mode cannot be used to create durable subscribers.

Also, queue receivers on a queue that is routed from another server are not permitted to specify `NO_ACKNOWLEDGE` mode.

EXPLICIT_CLIENT_ACKNOWLEDGE

`EXPLICIT_CLIENT_ACKNOWLEDGE` is like `CLIENT_ACKNOWLEDGE` except it acknowledges only the individual message, rather than all messages received so far on the session.

One example of when `EXPLICIT_CLIENT_ACKNOWLEDGE` would be used is when receiving messages and putting the information in a database. If the database insert operation is slow, you may want to use multiple application threads all doing simultaneous inserts. As each thread finishes its insert, it can use `EXPLICIT_CLIENT_ACKNOWLEDGE` to acknowledge only the message that it is currently working on.

`EXPLICIT_CLIENT_DUPS_OK_ACKNOWLEDGE`

`EXPLICIT_CLIENT_DUPS_OK_ACKNOWLEDGE` is like `DUPS_OK_ACKNOWLEDGE` except it "lazily" acknowledges only the individual message, rather than all messages received so far on the session.

Message Selectors

A message selector is a string that lets a client program specify a set of messages, based on the values of message headers and properties. A selector *matches* a message if, after substituting header and property values from the message into the selector string, the string evaluates to `true`. Consumers can request that the server deliver only those messages that match a selector.

The syntax of selectors is based on a subset of the SQL92 conditional expression syntax.

Identifiers

Identifiers can refer to the values of message headers and properties, but not to the message body. Identifiers are case-sensitive.

- Basic Syntax An identifier is a sequence of letters and digits, of any length, that begins with a letter. As in Java, the set of letters includes `_` (underscore) and `$` (dollar).
- Illegal Certain names are exceptions, which cannot be used as identifiers. In particular, `NULL`, `TRUE`, `FALSE`, `NOT`, `AND`, `OR`, `BETWEEN`, `LIKE`, `IN`, `IS`, and `ESCAPE` are defined to have special meaning in message selector syntax.
- Value Identifiers refer either to message header names or property names. The type of an identifier in a message selector corresponds to the type of the header or property value. If an identifier refers to a header or property that does not exist in a message, its value is `NULL`.

Literals

- String Literal A string literal is enclosed in single quotes. To represent a single quote within a literal, use two single quotes; for example, `'literal''s'`. String literals use the Unicode character encoding. String literals are case sensitive.
- Exact Numeric Literal An exact numeric literal is a numeric value without a decimal point, such as `57`, `-957`, and `+62`; numbers in the range of `long` are supported.
- Approximate Numeric Literal An approximate numeric literal is a numeric value with a decimal point (such as `7.`, `-95.7`, and `+6.2`), or a numeric value in scientific notation (such as `7E3` and `-57.9E2`); numbers in the range of `double` are supported. Approximate literals use the floating-point literal syntax of the Java programming language.

Boolean Literal	<p>The boolean literals are <code>TRUE</code> and <code>FALSE</code> (case insensitive).</p> <p>Internal computations of expression values use a 3-value boolean logic similar to SQL. However, the final value of an expression is always either <code>TRUE</code> or <code>FALSE</code>—never <code>UNKNOWN</code>.</p>
-----------------	---

Expressions

Selectors as Expressions	Every selector is a conditional expression. A selector that evaluates to <code>true</code> matches the message; a selector that evaluates to <code>false</code> or <code>unknown</code> does not match.
Arithmetic Expression	Arithmetic expressions are composed of numeric literals, identifiers (that evaluate to numeric literals), arithmetic operations, and smaller arithmetic expressions.
Conditional Expression	Conditional expressions are composed of comparison operations, logical operations, and smaller conditional expressions.
Order of Evaluation	Order of evaluation is left-to-right, within precedence levels. Parentheses override this order.

Operators

Case Insensitivity	Operator names are case-insensitive.
Logical Operators	Logical operators in precedence order: <code>NOT</code> , <code>AND</code> , <code>OR</code> .
Comparison Operators	<p>Comparison operators: <code>=</code>, <code>></code>, <code>>=</code>, <code><</code>, <code><=</code>, <code><></code> (not equal).</p> <p>These operators can compare only values of comparable types. (Exact numeric values and approximate numerical values are comparable types.) Attempting to compare incomparable types yields <code>false</code>. If either value in a comparison evaluates to <code>NULL</code>, then the result is unknown (in SQL 3-valued logic).</p> <p>Comparison of string values is restricted to <code>=</code> and <code><></code>. Two strings are equal if and only if they contain the same sequence of characters.</p> <p>Comparison of boolean values is restricted to <code>=</code> and <code><></code>.</p>
Arithmetic Operators	<p>Arithmetic operators in precedence order:</p> <ul style="list-style-type: none"> • <code>+</code>, <code>-</code> (unary) • <code>*</code>, <code>/</code> (multiplication and division) • <code>+</code>, <code>-</code> (addition and subtraction)

Arithmetic operations obey numeric promotion rules of the Java programming language.

Between Operator *arithmetic-expr1* [NOT] BETWEEN *arithmetic-expr2* AND *arithmetic-expr3*

The BETWEEN comparison operator includes its endpoints. For example:

- `age BETWEEN 5 AND 9` is equivalent to `age >= 5 AND age <= 9`
- `age NOT BETWEEN 5 AND 9` is equivalent to `age < 5 OR age > 9`

String Set Membership *identifier* [NOT] IN (*string-literal1*, *string-literal2*, ...)

The *identifier* must evaluate to either a string or NULL. If it is NULL, then the value of this expression is unknown.

Pattern Matching *identifier* [NOT] LIKE *pattern-value* [ESCAPE *escape-character*]

The *identifier* must evaluate to a string.

The *pattern-value* is a string literal, in which some characters bear special meaning:

- `_` (underscore) can match any single character.
- `%` (percent) can match any sequence of zero or more characters.
- *escape-character* preceding either of the special characters changes them into ordinary characters (which match only themselves).

Null Header or Property *identifier* IS NULL

This comparison operator tests whether a message header is null, or a message property is absent.

identifier IS NOT NULL

This comparison operator tests whether a message header or message property is non-null.

White Space

White space is any of the characters space, horizontal tab, form feed, or line terminator—or any contiguous run of characters in this set.

Data Type Conversion

[Table 7](#) summarizes legal datatype conversions. The symbol X in [Table 7](#) indicates that a value written into a message as the row type can be extracted as the column type. This table applies to all message values—including map pairs, headers and properties—except as noted below.

Table 7 Data Type Conversion

	bool	byte	short	char	int	long	float	double	string	byte[]
bool	X								X	
byte		X	X		X	X			X	
short			X		X	X			X	
char				X					X	
int					X	X			X	
long						X			X	
float							X	X	X	
double								X	X	
string	X	X	X		X	X	X	X	X	
byte[]										X

- Notes
- Message properties cannot have byte array values.
 - Values written as strings can be extracted as a numeric or boolean type only when it is possible to parse the string as a number of that type.

Sending Messages Synchronously and Asynchronously

TIBCO Enterprise Message Service supports two modes of sending messages:

- **Synchronous** sending blocks the application thread until the entire send is complete.
- **Asynchronous** sending offloads the notification of the success or failure to another thread, thereby increasing performance in certain situations.



Asynchronous sending is currently available only with the EMS client for Java.

Each sending mode has certain benefits. The following sections describe the benefits of the different modes.

Sending Synchronously

Because synchronous sending does not have the overhead involved in asynchronous sending, it yields better performance in most cases. Synchronous sending is also the best choice when sending the following types of messages:

- **Non-Persistent Messages** When high performance is a concern, use synchronous sending for non-persistent or reliable messages. Although asynchronous sending of non-persistent messages is supported, it is generally not recommended.
- **Transactions** Typically, it makes sense for applications to use synchronous sending when using transactions. Sending messages within a transaction does not require a response from the server, so higher throughput can be obtained sending synchronously within a transaction.

Synchronous sending simplifies a transaction; coordination of asynchronous send notifications and committing or rolling back a transaction introduces complexity to the application.

See [Sending Messages on page 354](#) for details.

Sending Asynchronously



Asynchronous sending is currently available only with the EMS client for Java.

The message producer can send messages asynchronously by registering a *completion listener* to monitor message send success or failure. Operating in a thread separate from that of the message producer, the completion listener manages the response to a successful or failed send, leaving the message producer free to perform other operations. See [Creating a Completion Listener for Asynchronous Sending on page 344](#) for details.

Asynchronous sending can increase performance in certain circumstances. One of the best uses for asynchronous sending is when sending persistent messages. High level outgoing message throughput can be obtained when sending non-transacted persistent messages.

There are other considerations for the application programmer when sending messages asynchronously. These considerations are described below.

Concurrent Message Use

For simplicity, it is suggested that application programmers create a new message for every asynchronous send call. If concurrent message use is acceptable in an application, messages may be reused when sending asynchronously, but generally it is *not* recommended due to the complexity it may add.



During asynchronous sends, the application programmer should be very aware of concurrent message usage between the application and the thread handling completion listeners. The message passed to the completion listener is the same message passed to the MessageProducer send method, which means modification of that particular message is reflected in both the application thread and the thread invoking the completion listener.

For example, if a TextMessage is asynchronously sent with the text of `foo`, and then the same message object's text is subsequently set to `bar`, it is conceivable that when the completion listener is invoked the message will contain `bar` even though it contained `foo` at the time it was sent.

Memory Use

Application programmers should be aware that some additional memory is used by the EMS server when asynchronously sending. Memory use increases if the performance of completion listeners is slower than overall application send rates.

Fault Tolerant Failovers

Because send notifications are handled in a separate thread when messages are sent asynchronously, it is possible to receive messages out of order after a fault tolerant switch.

For example, consider an application that sends messages A, B, and C. Message A succeeds, Message B fails, but message C succeeds immediately after reconnect to the fault tolerant server. The application may not know message B failed before message C was sent. Message consumers could conceivably receive messages in the order of A, C, B; it is up to the application to appropriately handle this situation.

Receiving Messages Synchronously and Asynchronously

The EMS APIs allow for both synchronous or asynchronous message consumption. For synchronous consumption, the message consumer explicitly invokes a receive call on the topic or queue. When synchronously receiving messages, the consumer remains blocked until a message arrives. See [Receiving Messages on page 355](#) for details.

The consumer can receive messages asynchronously by registering a *message listener* to receive the messages. When a message arrives at the destination, the message listener delivers the message to the message consumer. The message consumer is free to do other operations between messages. See [Creating a Message Listener for Asynchronous Message Consumption on page 348](#) for details.

Chapter 3 Destinations

This chapter describes destinations (topics and queues).

Topics

- [Destination Overview, page 52](#)
- [Destination Properties, page 58](#)
- [Creating and Modifying Destinations, page 75](#)
- [Wildcards, page 77](#)
- [Inheritance, page 80](#)
- [Destination Bridges, page 82](#)
- [Flow Control, page 87](#)
- [Delivery Delay, page 91](#)

Destination Overview

Destinations for messages can be either Topics or Queues. A destination can be created statically in the server configuration files, or dynamically by a client application.

Servers connected by routes exchange messages sent to temporary topics. As a result, temporary topics are ideal destinations for reply messages in request/reply interactions.

[Table 8](#) summarizes the differences between static, dynamic, and temporary destinations. The sections that follow provide more detail.

Table 8 Destination Overview (Sheet 1 of 2)

Aspect	Static	Dynamic	Temporary
Purpose	Static destinations let administrators configure EMS behavior at the enterprise level. Administrators define these administered objects, and client programs use them—relieving program developers and end users of the responsibility for correct configuration.	Dynamic destinations give client programs the flexibility to define destinations as needed for short-term use.	Temporary destinations are ideal for limited-scope uses, such as reply subjects.
Scope of Delivery	Static destinations support concurrent use. That is, several client processes (and in several threads within a process) can create local objects denoting the destination, and consume messages from it.	Dynamic destinations support concurrent use. That is, several client processes (and in several threads within a process) can create local objects denoting the destination, and consume messages from it.	Temporary destinations support only local use. That is, only the client connection that created a temporary destination can consume messages from it. However, servers connected by routes do exchange messages sent to temporary topics.
Creation	Administrators create static destinations using EMS server administration tools or API.	Client programs create dynamic destinations, if permitted by the server configuration.	Client programs create temporary destinations.

Table 8 Destination Overview (Sheet 2 of 2)

Aspect	Static	Dynamic	Temporary
Lookup	Client programs lookup static destinations by name. Successful lookup returns a local object representation of the destination.	Not applicable.	Not applicable.
Duration	A static destination remains in the server until an administrator explicitly deletes it.	<p>A dynamic destination remains in the server as long as at least one client actively uses it. The server automatically deletes it (at a convenient time) when all applicable conditions are true:</p> <ul style="list-style-type: none">• Topic or Queue all client programs that access the destination have disconnected• Topic no offline durable subscribers exist for the topic• Queue queue, no messages are stored in the queue	A temporary destination remains in the server either until the client that created it explicitly deletes it, or until the client disconnects from the server.

Destination Names

A destination name is a string divided into elements, each element separated by the dot character (.). The dot character allows you to create multi-part destination names that categorize destinations.

For example, you could have an accounting application that publishes messages on several destinations. The application could prefix all messages with ACCT, and each element of the name could specify a specific component of the application. ACCT.GEN_LEDGER.CASH, ACCT.GEN_LEDGER.RECEIVABLE, and ACCT.GEN_LEDGER.MISC could be subjects for the general ledger portion of the application.

Separating the subject name into elements allows applications to use wildcards for specifying more than one subject. See [Wildcards on page 77](#) for more information. The use of wildcards in destination names can also be used to define "parent" and "child" destination relationships, where the child destinations inherit the properties from its parents. See [Inheritance of Properties on page 80](#).

Static Destinations

Configuration information for static destinations is stored in configuration files for the EMS server. Changes to the configuration information can be made in a variety of ways. To manage static destinations, you can edit the configuration files using a text editor, you can use the administration tool, or you can use the administration APIs.

Clients can obtain references to static destinations through a naming service such as JNDI or LDAP. See [Creating and Modifying Destinations on page 75](#) for more information about how clients use static destinations.

Dynamic Destinations

Dynamic destinations are created on-the-fly by the EMS server, as required by client applications. Dynamic destinations do not appear in the configuration files and exist as long as there are messages or consumers on the destination. A client cannot use JNDI to lookup dynamic queues and topics.

When you use the [show queues](#) or [show topics](#) command in the administration tool, you see dynamic topics and queues have an asterisk (*) in front of their name in the list of topics or queues. If a property of a queue or topic has an asterisk (*) character in front of its name, it means that the property was inherited from the parent queue or topic and cannot be changed.

See [Dynamically Creating Topics and Queues on page 340](#) for details on topics and queues can be dynamically created by the EMS server.

Temporary Destinations

TIBCO Enterprise Message Service supports temporary destinations as defined in JMS specification and its API.

Servers connected by routes exchange messages sent to temporary topics. As a result, temporary topics are ideal destinations for reply messages in request/reply interactions.

For more information on temporary queues and topics, refer to the JMS documentation described in [Third Party Documentation on page xxix](#).

Destination Bridges

You can create server-based bridges between destinations of the same or different types to create a hybrid messaging model for your application. This allows all messages delivered to one destination to also be delivered to the bridged destination. You can bridge between different destination types, between the same destination type, or to more than one destination. For example, you can create a bridge between a topic and a queue or from a topic to another topic.

See [Destination Bridges on page 82](#) for more information about destination bridging.

Destination Name Syntax

TIBCO Enterprise Message Service places few restrictions on the syntax and interpretation of destination names. System designers and developers have the freedom to establish their own conventions when creating destination names. The best destination names reflect the structure of the data in the application itself.

Structure A destination name is a string divided into elements, each element separated by the dot character (.). The dot character allows you to create multi-part destination names that categorize destinations.

Empty strings ("") are not permitted destination names. Likewise, elements cannot incorporate the dot character by using an escape sequence.

Although they are not prohibited, we recommend that you do not use tabs, spaces, or any unprintable character in a destination name. You may, however, use wildcards. See [Wildcards on page 77](#) for more information.

Length Destinations are limited to a total length of 249 characters. However, some of that length is reserved for internal use. The amount of space reserved for internal use varies according to the number of elements in the destination; destinations that include the maximum number of elements are limited to 196 characters.

A destination can have up to 64 elements. Elements cannot exceed 127 characters. Dot separators are not included in element length.

- Destination Name Performance Considerations**
- When designing destination naming conventions, remember these performance considerations:
- Shorter destination names perform better than long destination names.
 - Destinations with several short elements perform better than one long element.
 - A set of destinations that differ early in their element lists perform better than subjects that differ only in the last element.

Special Characters in Destination Names These characters have special meanings when used in destination names:

Table 9 Characters with Special Meaning in Destination Names

Char	Char Name	Special Meaning
.	Dot	Separates elements within a destination name.
>	Greater-than	Wildcard character, matches one or more trailing elements.

Table 9 Characters with Special Meaning in Destination Names

Char	Char Name	Special Meaning
*	Asterisk	Wildcard character, matches one element.

For more information on wildcard matching, see [Wildcards * and > on page 77](#).

Examples

These examples illustrate the syntax for destination names.

Table 10 Valid Destination Name Examples

NEWS.LOCAL.POLITICS.CITY_COUNCIL
NEWS.NATIONAL.ARTS.MOVIES.REVIEWS
CHAT.MRKTG.NEW_PRODUCTS
CHAT.DEVELOPMENT.BIG_PROJECT.DESIGN
News.Sports.Baseball
finance
This.long.subject_name.is.valid.even.though.quite.uninformative

Table 11 Invalid Destination Name Examples

News..Natural_Disasters.Flood (null element)
WRONG. (null element)
.TRIPLE.WRONG.. (three null elements)
News.Tennis.Stats.Roger\Federer (backslash in the element Roger will be included in the element name, and will not escape the dot)

Destination Properties

This section contains a description of properties for topics and queues.

You can set the properties directly in the `topics.conf` or `queues.conf` file or by means of the `setprop topic` or `setprop queue` command in the EMS Administration Tool.

Table 12 lists the properties that can be assigned to topics and queues. The following sections describe each property.

Table 12 Destination properties (Sheet 1 of 2)

Property	Described on Page	Topic	Queue
<code>channel</code>	59	Yes	No
<code>exclusive</code>	59	No	Yes
<code>expiration</code>	60	Yes	Yes
<code>export</code>	61	Yes	No
<code>flowControl</code>	61	Yes	Yes
<code>global</code>	62	Yes	Yes
<code>import</code>	62	Yes	Yes
<code>maxbytes</code>	63	Yes	Yes
<code>maxmsgs</code>	64	Yes	Yes
<code>maxRedelivery</code>	64	No	Yes
<code>overflowPolicy</code>	65	Yes	Yes
<code>prefetch</code>	68	Yes	Yes
<code>redeliveryDelay</code>	70	No	Yes
<code>secure</code>	71	Yes	Yes
<code>sender_name</code>	72	Yes	Yes
<code>sender_name_enforced</code>	72	Yes	Yes
<code>store</code>	73	Yes	Yes

Table 12 Destination properties (Sheet 2 of 2)

Property	Described on Page	Topic	Queue
trace	74	Yes	Yes

channel

The `channel` property determines the multicast channel over which messages sent to this topic are broadcast. By including the `channel` property, the associated topic is enabled for multicast.

Set the `channel` property using this form:

```
channel=name
```

where *name* is the name of a channel, as defined in the `channels.conf` file.

For example, this will broadcast all messages sent to the topic `topic.foo` over the channel named `mycast`:

```
topic.foo channel=mycast
```

Only one channel is allowed for each topic. For this reason, overlapping wildcard topics are incompatible with channel properties. The creation of a wildcard topic with a `channel` property that overlaps with another wildcard topic with a `channel` property will fail. See [Overlapping Wildcards and Disjoint Properties on page 77](#) for more information.



This parameter cannot be used without first configuring multicast channels in the `channels.conf` file and enabling this feature in the `tibemsd.conf` file.

For more information, see [Chapter 13, Using Multicast, on page 369](#).

exclusive

The `exclusive` property is available for queues only (not for topics), and cannot be used with global queues.

When `exclusive` is set for a queue, the server sends all messages on that queue to one consumer. No other consumers can receive messages from the queue. Instead, these additional consumers act in a *standby* role; if the primary consumer fails, the server selects one of the standby consumers as the new primary, and begins delivering messages to it.

You can set `exclusive` using the form:

```
exclusive
```

Non-Exclusive Queues & Round-Robin Delivery By default, `exclusive` is not set for queues and the server distributes messages in a round-robin—one to each receiver that is ready. If any receivers are still ready to accept additional messages, the server distributes another round of messages—one to each receiver that is still ready. When none of the receivers are ready to receive more messages, the server waits until a queue receiver reports that it can accept a message.

This arrangement prevents a large buildup of messages at one receiver and balances the load of incoming messages across a set of queue receivers.

expiration

If an `expiration` property is set for a destination, the server honors the overridden expiration period and retains the message for the length of time specified by the `expiration` property.

However, the server overrides the `JMSExpiration` value set by the producer in the message header with the value 0 and therefore the consuming client does not expire the message.

You can set the `expiration` property for any queue and any topic using the form:

```
expiration=time[msec|sec|min|hour|day]
```

where *time* is the number of seconds. Zero is a special value that indicates messages to the destination never expire.

You can optionally include time units, such as `msec`, `sec`, `min`, `hour` or `day` to describe the *time* value as being in milliseconds, seconds, minutes, hours, or days, respectively. For example:

```
expiration=10min
```

Means 10 minutes.

When a message expires it is either destroyed or, if the `JMS_TIBCO_PRESERVE_UNDELIVERED` property on the message is set to `true`, the message is placed on the undelivered queue so it can be handled by a special consumer. See [Undelivered Message Queue on page 21](#) for details.

In EMS version 4.4 and later, clients automatically synchronize their clocks with the server when a connection is created. However, for long-lasting connections, Network Time Protocol (NTP) is the most reliable method for ensuring continuing synchronization between server and client. Additionally, if your EMS server or client application are based on a version of EMS prior to 4.4, you must ensure that clocks are synchronized among all the host computers that send and receive messages, if your pre-4.4 client application uses non-zero values for message expiration. Synchronize clocks to a tolerance that is a very small fraction of the smallest message expiration time.

export

The `export` property allows messages published by a client to a topic to be exported to the external systems with configured transports.

You can set `export` using the form:

```
export="list"
```

where *list* is one or more transport names, as specified by the `[transport_name]` ids in the `transports.conf` file. Multiple transport names in the list are separated by commas.

For example:

```
export="RV1,RV2"
```

Currently you can configure transports for SmartSockets or Rendezvous reliable and certified messaging protocols. You can specify the name of one or more transports of the same type in the `export` property.

You must purchase, install, and configure the external system (for example, Rendezvous) before configuring topics with the `export` property. Also, you must configure the communication parameters to the external system by creating a named transport in the `transports.conf` file.

For complete details about external message services, see these chapters:

- [Chapter 15, Working With TIBCO Rendezvous, on page 401](#)
- [Chapter 16, Working With TIBCO SmartSockets, on page 425](#)

flowControl

The `flowControl` property specifies the target maximum size the server can use to store pending messages for the destination. Should the number of messages exceed the maximum, the server will slow down the producers to the rate required by the message consumers. This is useful when message producers send messages much more quickly than message consumers can consume them. Unlike the behavior established by the `overflowPolicy` property, `flowControl` never discards messages or generates errors back to producer.

You can set `flowControl` using the form:

```
flowControl=size[KB|MB|GB]
```

where *size* is the maximum number of bytes of storage for pending messages of the destination. If you specify the `flowControl` property without a value, the target maximum is set to 256KB.

You can optionally include a KB, MB or GB after the number to specify kilobytes, megabytes, or gigabytes, respectively. For example:

```
flowControl=1000KB
```

Means 1000 kilobytes.

The `flow_control` parameter in `tibemsd.conf` file must be set to `enabled` before the value in this property is enforced by the server. See [Flow Control on page 87](#) for more information about flow control.

global

Messages destined for a topic or queue with the `global` property set are routed to the other servers that are participating in routing with this server.

You can set `global` using the form:

```
global
```

For further information on routing between servers, see [Chapter 20, Working With Routes, on page 509](#).

import

The `import` property allows messages published by an external system to be received by a EMS destination (a topic or a queue), as long as the transport to the external system is configured.

You can set `import` using the form:

```
import="list"
```

where *list* is one or more transport names, as specified by the `[NAME]` ids in the `transports.conf` file. Multiple transport names in the list are separated by commas. For example:

```
import="RV1,RV2"
```

Currently you can configure transports for TIBCO SmartSockets or TIBCO Rendezvous reliable and certified messaging protocols. You can specify the name of one or more transports of the same type in the `import` property.

You must purchase, install, and configure the external system (for example, Rendezvous) before configuring topics with the `import` property. Also, you must configure the communication parameters to the external system by creating a named transport in the `transports.conf` file.

For complete details about external message services, see these chapters:

- [Chapter 15, Working With TIBCO Rendezvous, on page 401](#)
- [Chapter 16, Working With TIBCO SmartSockets, on page 425](#)

maxbytes

Topics and queues can specify the `maxbytes` property in the form:

```
maxbytes=value[KB|MB|GB]
```

where *value* is the number of bytes. For example:

```
maxbytes=1000
```

Means 1000 bytes.

You can optionally include a KB, MB or GB after the number to specify kilobytes, megabytes, or gigabytes, respectively. For example:

```
maxbytes=1000KB
```

Means 1000 kilobytes.

For queues, `maxbytes` defines the maximum size (in bytes) that the queue can store, summed over all messages in the queue. Should this limit be exceeded, messages will be rejected by the server and the message producer send calls will return an error (see also [overflowPolicy](#)). For example, if a receiver is off-line for a long time, then the queue size could reach this limit, which would prevent further memory allocation for additional messages.

If `maxbytes` is zero, or is not set, the server does not limit the memory allocation for the queue.

You can set both `maxmsgs` and `maxbytes` properties on the same queue. Exceeding either limit causes the server to reject new messages until consumers reduce the queue size to below these limits.

For topics, `maxbytes` limits the maximum size (in bytes) that the topic can store for delivery to each durable or non-durable online subscriber on that topic. That is, the limit applies separately to each subscriber on the topic. For example, if a durable subscriber is off-line for a long time, pending messages accumulate until they exceed `maxbytes`; when the subscriber consumes messages (freeing storage) the topic can accept additional messages for the subscriber. For a non-durable subscriber, `maxbytes` limits the number of pending messages that can accumulate while the subscriber is online.



Under certain conditions, because of the pipelined nature of message processing or the requirements of transactional messaging, the `maxbytes` limit can be slightly exceeded. You may see message totals that are marginally larger than the set limit.

When a destination inherits different values of this property from several parent destinations, it inherits the smallest value.



You can further protect against consumers that receive messages without acknowledging them using the parameter `disconnect_non_acking_consumers`.

maxmsgs

Topics and queues can specify the `maxmsgs` property in the form:

`maxmsgs=value`

where *value* defines the maximum number of messages that can be waiting in a queue. When adding a message would exceed this limit, the server does not accept the message into storage, and the message producer's `send` call returns an error (but see also [overflowPolicy](#)).

If `maxmsgs` is zero, or is not set, the server does not limit the number of messages in the queue.

You can set both `maxmsgs` and `maxbytes` properties on the same queue. Exceeding either limit causes the server to reject new messages until consumers reduce the queue size to below these limits.



Under certain conditions, because of the pipelined nature of message processing or the requirements of transactional messaging, the `maxmsgs` limit can be slightly exceeded. You may see message totals that are marginally larger than the set limit.



You can further protect against consumers that receive messages without acknowledging them using the parameter [disconnect_non_acking_consumers](#).

maxRedelivery

The `maxRedelivery` property specifies the number of attempts the server should make to deliver a message sent to a queue. Set `maxRedelivery` using the form:

`maxRedelivery=count`

where *count* is an integer between 2 and 255 that specifies the maximum number of times a message can be delivered to receivers. A value of zero disables `maxRedelivery`, so there is no maximum.

Once the server has attempted to deliver the message the specified number of times, the message is either destroyed or, if the [JMS_TIBCO_PRESERVE_UNDELIVERED](#) property on the message is set to `true`, the message is placed on the undelivered queue so it can be handled by a special consumer. See [Undelivered Message Queue on page 21](#) for details.

For messages that have been redelivered, the `JMSRedelivered` header property is set to `true` and the `JMSXDeliveryCount` property is set to the number of times the message has been delivered to the queue. If the server restarts, the current number of delivery attempts in the `JMSXDeliveryCount` property is not retained.



In the event of an abrupt exit by the client, the `maxRedelivery` count can be mistakenly incremented. An abrupt exit prevents the client from communicating with the server; for example, when the client exits without closing the connection or when the client application crashes. If a client application exits abruptly, the EMS server counts all messages sent to the client as delivered, even if they were not presented to the application.

For more information, see [Undelivered Message Queue on page 21](#).

overflowPolicy

Topics and queues can specify the `overflowPolicy` property to change the effect of exceeding the message capacity established by either `maxbytes` or `maxmsgs`.

Set the `overflowPolicy` using the form:

```
overflowPolicy=default|discardOld|rejectIncoming
```

If `overflowPolicy` is not set, then the policy is `default`.

The effect of `overflowPolicy` differs depending on whether you set it on a topic or a queue, so the impact of each `overflowPolicy` value is described separately for topics and queues.

When `overflowPolicy` is set on multicast-enabled topics, it is honored in the multicast daemon. That is, the multicast daemon will discard messages based on the backlog in the daemon rather than the backlog in the server.

For topics and consumers that are not multicast-enabled, the response to the `overflowPolicy` occurs in the EMS server.

If wildcards are used in the `.conf` file the inheritance of the `overflowPolicy` policy from multiple parents works as follows:

- If a child destination has a non-default `overflowPolicy` policy set, then that policy is used and it does not inherit any conflicting policy from a parent.
- If a parent has `OVERFLOW_REJECT_INCOMING` set, then it is inherited by the child destination over any other policy.
- If no parent has `OVERFLOW_REJECT_INCOMING` set and a parent has `OVERFLOW_DISCARD_OLD` policy set, then that policy is inherited by the child destination.
- If no parent has the `OVERFLOW_REJECT_INCOMING` or `OVERFLOW_DISCARD_OLD` set, then the `default` policy is used by the child destination.

default

For topics, `default` specifies that messages are sent to each subscriber in turn. If the `maxbytes` or `maxmsgs` setting has been reached for a subscriber, that subscriber does not receive the message. No error is returned to the message producer.

For queues, `default` specifies that new messages are rejected by the server and an error is returned to the producer if the established `maxbytes` or `maxmsgs` value has been exceeded.



When delivery delay is enabled for a topic, the behavior of `overflowPolicy=default` mimics that of a queue. That is, when `maxbytes` or `maxmsgs` has been reached, new messages are rejected by the server and an error is returned to the producer.

Note that this is the same default behavior for topics and queues as in EMS 4.3.

discardOld

For topics, `discardOld` specifies that, if any of the subscribers have an outstanding number of undelivered messages on the server that are over the message limit, the oldest messages are discarded before they are delivered to the subscriber.

The `discardOld` setting impacts subscribers individually. For example, you might have three subscribers to a topic, but only one subscriber exceeds the message limit. In this case, only the oldest messages for the one subscriber are discarded, while the other two subscribers continue to receive all of their messages.

When messages for a topic or queue exceed the `maxbytes` or `maxmsgs` value, the oldest messages are silently discarded. No error is returned to the producer.

rejectIncoming

For topics, `rejectIncoming` specifies that, if *any* of the subscribers have an outstanding number of undelivered messages on the server that are over the message limit, all new messages are rejected and an error is returned to the producer.

For queues, `rejectIncoming` specifies that, if messages on the queue have exceeded the `maxbytes` or `maxmsgs` value, all new messages are rejected and an error is returned to the producer. (This is the same as the `default` behavior.)

Examples

To discard messages on `myQueue` when the number of queued messages exceeds 1000, enter:

```
setprop queue myQueue maxmsgs=1000,overflowPolicy=discardOld
```

To reject all new messages published to `myTopic` when the memory used by undelivered messages for any of the topic subscribers exceeds 100KB, enter:

```
setprop topic myTopic maxbytes=100KB,overflowPolicy=rejectIncoming
```

prefetch

The message consumer portion of a client and the server cooperate to regulate fetching according to the `prefetch` property. The `prefetch` property applies to both topics and queues.

You can set `prefetch` using the form:

```
prefetch=value
```

where *value* is one of the values in [Table 13](#).

Table 13 Prefetch

Value	Description
2 or more	<p>The message consumer automatically fetches messages from the server. The message consumer never fetches more than the number of messages specified by <i>value</i>.</p> <p>See Automatic Fetch Enabled on page 69 for details.</p>
1	<p>The message consumer automatically fetches messages from the server—initiating fetch only when it does not currently hold a message.</p>
none	<p>Disables automatic fetch. That is, the message consumer initiates fetch only when the client calls <code>receive</code>—either an explicit synchronous call, or an implicit call (in an asynchronous consumer).</p> <p>This value cannot be used with topics or global queues.</p> <p>See Automatic Fetch Disabled on page 70 for details.</p>
0	<p>The destination inherits the <code>prefetch</code> value from a parent destination with a matching name. If it has no parent, or no destination in the parent chain sets a value for <code>prefetch</code>, then the default value is 5 queues and 64 for topics.</p> <p>When a destination does not set any value for <code>prefetch</code>, then the default value is 0 (zero; that is, inherit the <code>prefetch</code> value).</p> <p>See Inheritance on page 70 for details.</p>



If both `prefetch` and `maxRedelivery` are set to a non-zero value, then there is a potential to lose prefetched messages if one of the messages exceeds the `maxRedelivery` limit. For example, `prefetch=5` and `maxRedelivery=4`. The first message is redelivered 4 times, hits the `maxRedelivery` limit and is sent to the undelivered queue (as expected). However, the other 4 pre-fetched messages are also sent to the undelivered queue and are not processed by the receiving application. The work around is to set `prefetch=none`, but this can have performance implications on large volume interfaces.

Background

Delivering messages from the server destination to a message consumer involves two independent phases—fetch and accept:

- The *fetch* phase is a two-step interaction between a message consumer and the server.
 - The message consumer initiates the fetch phase by signalling to the server that it is ready for more messages.
 - The server responds by transferring one or more messages to the client, which stores them in the message consumer.
- In the *accept* phase, client code takes a message from the message consumer.

The `receive` call embraces both of these phases. It initiates fetch when needed and it accepts a message from the message consumer.

To reduce waiting time for client programs, the message consumer can *prefetch* messages—that is, fetch a batch of messages from the server, and hold them for client code to accept, one by one.

Automatic Fetch Enabled

To enable automatic fetch, set `prefetch` to a positive integer. Automatic fetch ensures that if a message is available, then it is waiting when client code is ready to accept one. It can improve performance by decreasing or eliminating client idle time while the server transfers a message.

However, when a queue consumer prefetches a group of messages, the server does not deliver them to other queue consumers (unless the first queue consumer's connection to the server is broken).



A positive `prefetch` must be configured in order to use `receiveNoWait` function calls.

Automatic Fetch Disabled

To disable automatic fetch, set `prefetch=none`.

Even when `prefetch=none`, a queue consumer can still hold a message. For example, a `receive` call initiates fetch, but its timeout elapses before the server finishes transferring the message. This situation leaves a fetched message waiting in the message consumer. A second `receive` call does not fetch another message; instead, it accepts the message that is already waiting. A third `receive` call initiates another fetch.

Notice that a waiting message still belongs to the queue consumer, and the server does not deliver it to another queue consumer (unless the first queue consumer's connection to the server is broken). To prevent messages from waiting in this state for long periods of time, code programs either to call `receive` with no timeout, or to call it (with timeout) repeatedly and shorten the interval between calls.



Automatic fetch cannot be disabled for global queues or for topics.

Inheritance

When a destination inherits the `prefetch` property from parent destination with matching names, these behaviors are possible:

- When all parent destinations set the value `none`, then the child destination inherits the value `none`.
- When any parent destination sets a non-zero numeric value, then the child destination inherits the *largest* value from among the entire parent chain.
- When none of the parent destinations sets any non-zero numeric value, then the child destination uses the default value (which is 5).

redeliveryDelay

When `redeliveryDelay` is set, the EMS server waits the specified interval before returning an unacknowledged message to the queue. When a previously delivered message did not receive a successful acknowledgement, the EMS server waits the specified redelivery delay before making the message available again in the queue. This is most likely to occur in the event of a transaction rollback, session or message recovery, session or connection close, or the abrupt exit of a client application. However, note that the delay time is not exact, and in most situations will exceed the specified `redeliveryDelay`.



The redelivery delay is not available for routed queues.

The value can be specified in seconds, minutes, or hours. The value may be in the range of 15 seconds and 8 hours.

You can set `redeliveryDelay` using the form:

```
redeliveryDelay=time[sec|min|hour]
```

where *time* is the number of seconds. Zero is a special value that indicates no redelivery delay.

You can optionally include time units, such as `sec`, `min`, or `hour` describe the *time* value as being in seconds, minutes, or hours, respectively. For example:

```
redeliveryDelay=30min
```

specifies a redelivery delay of 30 minutes.

During the delay interval, messages are placed in the `$sys.redelivery.delay` queue. This queue can be browsed, but it cannot be consumed from or purged. However, purging the queue from which the delayed message came, or removing the message using its message ID, immediately removes that message from `$sys.redelivery.delay`.



While a message is on the `$sys.redelivery.delay` queue, it is not on the queue from which it came and so it is not included in statistical message counts. This includes `maxmsgs`, `maxbytes`, `flowControl`, and so on.

secure

When the `secure` property is enabled for a destination, it instructs the server to check user permissions whenever a user attempts to perform an operation on that destination.

You can set `secure` using the form:

```
secure
```

If the `secure` property is not set for a destination, the server does not check permissions for that destination and any authenticated user can perform any operation on that topic or queue.



The `secure` property is independent of SSL—it controls basic authentication and permission verification within the server. To configure secure communication between clients and server, see [Using the SSL Protocol on page 465](#).

The server `authorization` property acts as a master switch for checking permissions. That is, the server checks user permissions on secure destinations only when the `authorization` property is enabled. To enforce permissions, you must *both* enable the `authorization` configuration parameter, and set the `secure` property on each affected destination.

See [Chapter 8, Authentication and Permissions, on page 265](#) for more information on permissions and the `secure` property.

sender_name

The `sender_name` property specifies that the server may include the sender's username for messages sent to this destination.

You can set `sender_name` using the form:

```
sender_name
```

When the `sender_name` property is enabled, the server takes the user name supplied by the message producer when the connection is established and places that user name into the `JMS_TIBCO_SENDER` property in the message.

The message producer can override this behavior by specifying a property on a message. If a message producer sets the `JMS_TIBCO_DISABLE_SENDER` property to true for a message, the server overrides the `sender_name` property and does not add the sender name to the message.

If authentication for the server is turned off, the server places whatever user name the message producer supplied when the message producer created a connection to the server. If authentication for the server is enabled, the server authenticates the user name supplied by the connection and the user name placed in the message property will be an authenticated user. If SSL is used, the SSL connection protocol guarantees the client is authenticated using the client's digital certificate.

sender_name_enforced

The `sender_name_enforced` property specifies that messages sent to this destination *must* include the sender's user name. The server retrieves the user name of the message producer using the same procedure described in the `sender_name` property above. However, unlike, the `sender_name` property, there is no way for message producers to override this property.

You can set `sender_name_enforced` using the form:

```
sender_name_enforced
```

If the `sender_name` property is also set on the destination, this property overrides the `sender_name` property.



In some business situations, clients may not be willing to disclose the username of their message producers. If this is the case, these clients may wish to avoid sending messages to destinations that have the `sender_name` or `sender_name_enforced` properties enabled.

In these situations, the operator of the EMS server should develop a policy for disclosing a list of destinations that have these properties enabled. This will allow clients to avoid sending messages to destinations that would cause their message producer usernames to be exposed.

store

The `store` property determines where messages sent to this destination are stored. Messages may be stored in a file, or in a database. See [Store Messages in Multiple Stores on page 30](#) for more information on using and configuring multiple stores.



When using the `setprop` or `addprop` commands to change the store settings for a topic or queue, note that existing messages are not migrated to the new store. As a result, stopping the EMS server and deleting the original store may result in data loss, if a destination still had messages in the original store.

Set the `store` property using this form:

```
store=name
```

where *name* is the name of a store, as defined in the `stores.conf` file.

For example, this will send all messages sent to the destination `giants.games` to the store named `baseball`; messages sent to all other destinations will be stored in `everythingelse`:

```
> store=everythingelse
giants.games store=baseball
```

Only one store is allowed for each destination. If there is a conflict, for example if overlapping wildcards cause a topic to inherit multiple store properties, the topic creation will fail.



This parameter cannot be used without first enabling this feature in the `tibemspd.conf` file. The `stores.conf` file must also exist, but can be left empty if the only store names that are associated with destinations are the default store files.

See [Store Messages in Multiple Stores on page 30](#) for more information.

trace

The `trace` property specifies that tracing should be enabled for this destination.

You can set `trace` using the form:

```
trace = [body]
```

Specifying `trace` (without `=body`), generates trace messages that include the message sequence, message ID, and message size. Specifying `trace=body` generates trace messages that include the message body. See [Message Tracing on page 452](#) for more information about message tracing.

Creating and Modifying Destinations

Destinations are typically "static" administered objects that can be stored in a JNDI or LDAP server. Administered objects can also be stored in the EMS server and looked up using the EMS implementation of JNDI. This section describes how to use the EMS Administration Tool described in [Chapter 6](#) to create and modify destination objects in EMS.

You create a queue using the `create queue` command and a topic using the `create topic` command. For example, to create a new queue named `myQueue`, enter:

```
create queue myQueue
```

To create a topic named `myTopic`, enter:

```
create topic myTopic
```

The queue and topic data stored on the EMS server is located in the `queues.conf` and `topics.conf` files, respectively. You can use the `show queues` and `show topics` commands to list all of the queues and topics on your EMS server and the `show queue` and `show topic` commands to show the configuration details of specific queues and topics.

A queue or topic may include optional properties that define the specific characteristics of the destination. These properties are described in [Destination Properties on page 58](#) and they can be specified when creating the queue or topic or modified for an existing queue or topic using the `addprop queue`, `addprop topic`, `setprop queue`, `setprop topic`, `removeprop queue`, and `removeprop topic` commands.

For example, to discard messages on `myQueue` when the number of queued messages exceeds 1000, you can set an `overflowPolicy` by entering:

```
addprop queue myQueue maxmsgs=1000,overflowPolicy=discardOld
```

To change the `overflowPolicy` from `discardOld` to `rejectIncoming`, enter:

```
addprop queue myQueue overflowPolicy=rejectIncoming
```

The `setprop queue` and `setprop topic` commands remove properties that are not explicitly set by the command. For example, to change `maxmsgs` to 100 and to `remove` the `overflowPolicy` parameter, enter:

```
setprop queue myQueue maxmsgs=100
```

Creating Secure Destinations

By default, all authenticated EMS users have permissions to perform any action on any topic or queue. You can set the [secure](#) property on a topic or queue and then use the [grant topic](#) or [grant queue](#) command to specify which users and/or groups are allowed to perform which actions on the destination.

The [secure](#) property requires that you enable the [authorization](#) property on the EMS server.

For example, to create a secure queue, named `myQueue`, to which only users "joe" and "eric" can send messages and "sally" can receive messages, in the EMS Administration Tool, enter:

```
set server authorization=enabled
create queue myQueue secure
grant queue myQueue joe send
grant queue myQueue eric send
grant queue myQueue sally receive
```

See [Chapter 8, Authentication and Permissions, on page 265](#) for more information.

Wildcards

You can use wildcards when specifying statically created destinations in `queues.conf` and `topics.conf`. The use of wildcards in destination names can be used to define "parent" and "child" destination relationships, where the child destinations inherit the properties and permissions from its parents. You must first understand wildcards to understand the inheritance rules described in [Inheritance on page 80](#).

Wildcards * and >

To understand the rules for inheritance of properties, it is important to understand the use of the two wildcards, * and >.

- The wildcard > by itself matches any destination name.
- When > is mixed with text, it matches one or more trailing elements. For example:

```
foo.>
```

Matches `foo.bar`, `foo.boo`, `foo.boo.bar`, and `foo.bar.boo`.

- The wildcard * means that any token can be in the place of *. For example:

```
foo.*
```

Matches `foo.bar` and `foo.boo`, but not `foo.bar.boo`.

```
foo.*.bar
```

Matches `foo.boo.bar`, but not `foo.bar`.

Overlapping Wildcards and Disjoint Properties

Some destination properties are disjoint, and the server allows that property to be set only once for each destination. If an existing destination includes a value for a disjoint property and you attempt to assign a different value, the action will fail.

Overlapping wildcard destinations can cause conflicts with disjoint properties. For example, consider the following configuration of the `channel` property:

```
topic.multicast.>          channel=multicast_1
topic.multicast.quotes.* channel=multicast_2
```

The topic `topic.multicast.quotes.tibx` would be assigned both channels, `multicast_1` and `multicast_2`. Therefore, the wildcard topics `topic.multicast.>` and `topic.multicast.quotes.*` cannot coexist. Their creation would fail.

The disjoint destination properties are:

- [channel](#)
- [store](#)

Wildcards in Topics

TIBCO Enterprise Message Service enables you to use wildcards in topic names in some situations:

- You can subscribe to wildcard topics.

If you subscribe to a topic containing a wildcard, you will receive any message published to a matching topic. For example, if you subscribe to `foo.*` you will receive messages published to a topic named `foo.bar`.

You can subscribe to a wildcard topic (for example `foo.*`), whether or not there is a matching topic in the configuration file (for example, `foo.*`, `foo.>`, or `foo.bar`). However, if there is no matching topic name in the configuration file, no messages will be published on that topic.

- You cannot publish to wildcard topics.
- If `foo.bar` is not in the configuration file, then you can publish to `foo.bar` if `foo.*` or `foo.>` exists in the configuration file.
- On routed topic messages, subscribers must specify a topic that is a direct subset (or equal) of the configured global topic. For more information, see [Wildcards on page 527](#).

Wildcards in Queues

TIBCO Enterprise Message Service enables you to use wildcards in queue names in some situations. You can neither send to nor receive from wildcard queue names. However, you can use wildcard queue names in the configuration files.

For example, if the queue configuration file includes a line:

```
foo.*
```

then users can dynamically create queues `foo.bar`, `foo.bob`, and so forth, but not `foo.bar.bob`.

Wildcards and Multicast

Messages published to multicast-enabled topics are sent on the multicast channels defined for those topics. A wildcard may cover multiple multicast-enabled topics, each on a different multicast channel.

For example, consider the following configuration:

```
>
topic.info
topic.quotes
topic.multicast.info      channel=channel-1
topic.multicast.quotes    channel=channel-2
```

A message consumer subscribed to `topic >` will receive messages published to `topic.info` and `topic.quotes` over TCP, will receive messages published to `topic.multicast.info` over the multicast channel `channel-1` and messages published to `topic.multicast.quotes` over the multicast channel `channel-2`. Note that this means a wildcard consumer may receive messages over both multicast and TCP.

Wildcards and Dynamically Created Destinations

As described in [Dynamically Creating Topics and Queues on page 340](#), the EMS server may dynamically create destinations on behalf of its clients. The use of wildcards in the `.conf` files can be used to control the allowable names of dynamically created destinations.

The same basic wildcard rules apply to dynamically created destinations as described above for static destinations.

Examples

- If the `queues.conf` file contains:

```
>
```

The EMS server can dynamically create a queue with any name.

- If the `topics.conf` file contains only:

```
foo.>
```

The EMS server can dynamically create topics with names like `foo.bar`, `foo.boo`, `foo.boo.bar`, and `foo.bar.boo`.

- If the `queues.conf` file contains only:

```
foo.*
```

The EMS server can dynamically create queues with names like `foo.bar` and `foo.boo`, but not `foo.bar.boo`.

- If the `topics.conf` file contains only:

```
foo.*.bar
```

The EMS server can dynamically create topics with names like `foo.boo.bar`, but not `foo.bar`.

Inheritance

This section describes the inheritance of properties and permissions. For more information on wildcards, refer to [Wildcards on page 77](#). For more information on destination properties, refer to [Destination Properties on page 58](#). For more information on permissions, refer to [Chapter 8, Authentication and Permissions, on page 265](#).

Inheritance of Properties

All destination properties are inheritable for both topics and queues. This means that a property set for a "wildcarded" destination is inherited by all destinations with matching names.

For example, if you have the following in your `topics.conf` file:

```
foo.* secure
foo.bar
foo.bob
```

Topics `foo.bar` and `foo.bob` are [secure](#) topics because they inherit `secure` from their parent, `foo.*`. If your EMS server were to dynamically create a `foo.new` topic, it too would have the `secure` property.

The properties inherited from a parent are *in addition* to the properties defined for the child destination.

For example, if you have the following in your `topics.conf` file:

```
foo.* secure
foo.bar sender_name
```

Then `foo.bar` has both the [secure](#) and [sender_name](#) properties.

In the above example, there is no way to make topic `foo.*` `secure` without making `foo.bar` `secure`. In other words, EMS does not offer the ability to remove inherited properties. However, for properties that are assigned values, you can override the value established in a parent.

For example, if you have the following in your `queues.conf` file:

```
foo.* maxbytes=200
foo.bar maxbytes=2000
```

The `foo.bar` queue has a [maxbytes](#) value of 2000.

When there are multiple ancestors for a destination, the destination inherits the properties from all of the parents. For example:

```
> sender_name
foo.* secure
foo.bar trace
```

The `foo.bar` topic has the `sender_name`, `secure` and `trace` properties.

When there are multiple parents for a destination that contain conflicting property values, the destination inherits the smallest value. For example:

```
> maxbytes=2000
foo.* maxbytes=200
foo.bar
```

The `foo.bar` topic has a `maxbytes` value of 200.

Property inheritance is powerful, but can be complex to understand and administer. You must plan before assigning properties to topics and queues. Using wildcards to assign properties must be used carefully. For example, if you enter the following line in the `topics.conf` file:

```
> store=mystore
```

you make every topic store messages, regardless of additional entries. This might require a great deal of memory for storage and greatly decrease the system performance.

Inheritance of Permissions

Inheritance of permissions is similar to inheritance of properties. If the parent has a permission, then the child inherits that permission. For example, if Bob belongs to GroupA, and GroupA has `publish` permission on a topic, then Bob has `publish` permission on that topic.

Permissions for a single user are the union of the permissions set for that user, and of all permissions set for every group in which the user is a member. These permission sets are additive. Permissions have positive boolean inheritance. Once a permission right has been granted through inheritance, it can not be removed.

All rules for wildcards apply to inheritance of permissions. For example, if a user has permission to publish on topic `foo.*`, the user also has permission to publish on `foo.bar` and `foo.new`.

For more information on wildcards, refer to [Wildcards on page 77](#). For more information on permissions, refer to [User Permissions on page 283](#).

Destination Bridges

Some applications require the same message to be sent to more than one destination, possibly of different types. For example, an application may send messages to a queue for distributed load balancing. That same application, however, may also need the messages to be published to several monitoring applications. Another example is an application that publishes messages to several topics. All messages however, must also be sent to a database for backup and for data mining. A queue is used to collect all messages and send them to the database.

An application can process messages so that they are sent multiple times to the required destinations. However, such processing requires significant coding effort in the application. EMS provides a server-based solution to this problem. You can create bridges between destinations so that messages sent to one destination are also delivered to all bridged destinations.

Bridges are created between one destination and one or more other destinations of the same or of different types. That is, you can create a bridge from a topic to a queue or from a queue to a topic. You can also create a bridge between one destination and multiple destinations. For example, you can create a bridge from topic `a.b` to queue `q.b` and topic `a.c`.

When specifying a bridge, you can specify a particular destination name, or you can use wildcards. For example, if you specify a bridge on topic `foo.*` to queue `foo.queue`, messages delivered to any topic matching `foo.*` are sent to `foo.queue`.



Because global topics are routed between servers and global queues are limited to their neighbors, in most cases the best practice is to send messages to a topic and then bridge the topic to a queue.

When multiple bridges exist, using wildcards to specify a destination name may result in a message being delivered twice. For example, if the queues `Q.1` and `Q.>` are both bridged to `QX.1`, the server will deliver two copies of sent messages to `QX.1`.

The following three figures illustrate example bridging scenarios.

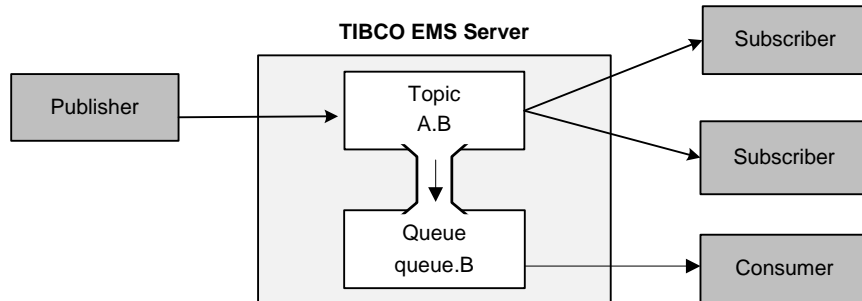
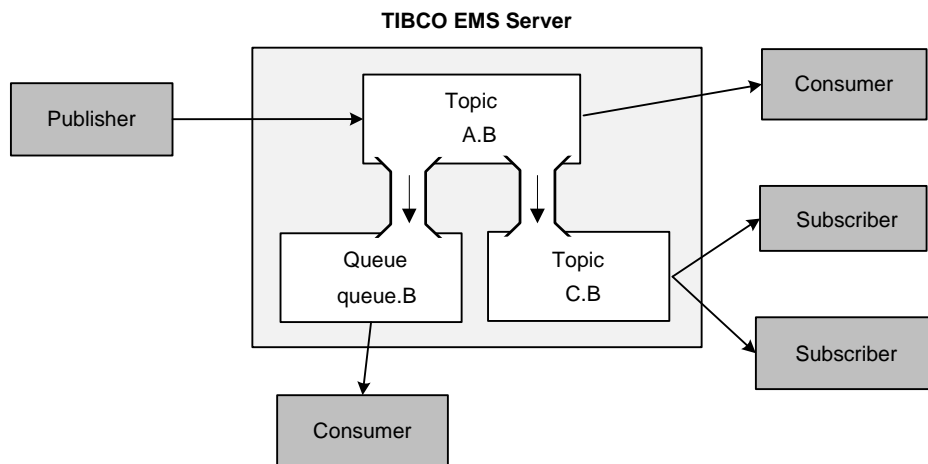
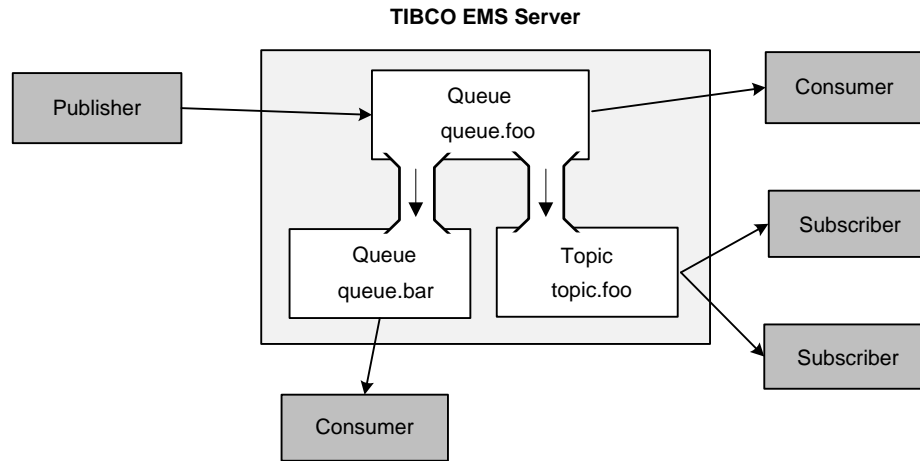
Figure 11 Bridging a topic to a queue*Figure 12 Bridging a topic to multiple destinations*

Figure 13 Bridging a queue to multiple destinations



When a bridge exists between two queues, the message is delivered to both queues. The queues operate independently; if the message is retrieved from one queue, that has no effect on the status of the message in the second queue.

Bridges are not transitive. That is, messages sent to a destination with a bridge are only delivered to the specified bridged destinations and are not delivered across multiple bridges. For example, topic A.B has a bridge to queue Q.B. Queue Q.B has a bridge to topic B.C. Messages delivered to A.B are also delivered to Q.B, but not to B.C.

The bridge copies the source message to the target destination, which assigns the copied message a new message identifier. Note that additional storage may be required, depending on the target destination store parameters.

Creating a Bridge

Bridges are configured in the `bridges.conf` configuration file. You specify a bridge using the following syntax:

```
[destinationType:destinationName]
    destinationType=destinationToBridgeTo selector="messageSelector"
```

where *destinationType* is the type of the destination (either `topic` or `queue`), *destinationName* is the name of the destination from which you wish to create a bridge, *destinationToBridgeTo* is the name of the destination you wish to create a bridge to, and `selector="messageSelector"` is an optional message selector to specify the subset of messages the destination should receive.

Each *destinationName* can specify wildcards, and therefore any destination matching the pattern will have the specified bridge. Each *destinationName* can specify more than one *destinationToBridgeTo*.

For example, the bridge illustrated in [Figure 11](#) and [Figure 12](#) would be specified as the following in `bridges.conf`:

```
[topic:A.B]
    queue=queue.B
    topic=C.B
```

Specifying a message selector on a bridged destination is described in the following section.

Selecting the Messages to Bridge

By default, all messages sent to a destination with a bridge are sent to all bridged destinations. This can cause unnecessary network traffic if each bridged destination is only interested in a subset of the messages sent to the original destination. You can optionally specify a message selector for each bridge to determine which messages are sent over that bridge.

Message selectors for bridged destinations are specified as the `selector` property on the bridge. The following is an example of specifying a selector on the bridges defined in the previous section:

```
[topic:A.B]
    queue=queue.B
    topic=C.B selector="urgency in('medium', 'high')"
```

For detailed information about message selector syntax, see the documentation for the `Message` class in the relevant EMS API reference document.

Access Control and Bridges

Message producers must have access to a destination to send messages to that destination. However, a bridge automatically has permission to send to its target destination. Special configuration is not required.

Transactions

When a message producer sends a message within a transaction, all messages sent across a bridge are part of the transaction. Therefore, if the transaction succeeds, all messages are delivered to all bridged destinations. If the transaction fails, no consumers for bridged destinations receive the messages.

If a message cannot be delivered to a bridged destination because the message producer does not have the correct permissions for the bridged destination, the transaction cannot complete, and therefore fails and is rolled back.

Flow Control

In some situations, message producers may send messages more rapidly than message consumers can receive them. The pending messages for a destination are stored by the server until they can be delivered, and the server can potentially exhaust its storage capacity if the message consumers do not receive messages quickly enough. To avoid this, EMS allows you to control the flow of messages to a destination. Each destination can specify a target maximum size for storing pending messages. When the target is reached, EMS blocks message producers when new messages are sent. This effectively slows down message producers until the message consumers can receive the pending messages.

Enabling Flow Control

The `flow_control` parameter in `tibemspd.conf` enables and disables flow control globally for the EMS server. When `flow_control` is disabled (the default setting), the server does not enforce any flow control on destinations. When `flow_control` is enabled, the server enforces any flow control settings specified for each destination. See [Chapter 7, Using the Configuration Files, on page 185](#) for more information about working with configuration parameters.

When you wish to control the flow of messages on a destination, set the `flowControl` property on that destination. The `flowControl` property specifies the target maximum size of stored pending messages for the destination. The size specified is in bytes, unless you specify the units for the size. You can specify KB, MB, or GB for the units. For example, `flowControl = 60MB` specifies the target maximum storage for pending messages for a destination is 60 Megabytes.

Enforcing Flow Control

The value specified for the `flowControl` property on a destination is a target maximum for pending message storage. When flow control is enabled, the server may use slightly more or less storage before enforcing flow control, depending upon message size, number of message producers, and other factors. Setting the `flowControl` property on a destination but specifying no value causes the server to use a default value of 256KB.

When the storage for pending messages is near the specified limit, the server blocks all new calls to send a message from message producers. The calls do not return until the storage has decreased below the specified limit, or the `flowControl` limit is increased. Once message consumers have received messages and the pending message storage goes below the specified limit, the server allows the send message calls to return to the caller and the message producers can continue processing.

If there are no message consumers for a destination, the server does not enforce flow control for the destination. That is, if a queue has no started receivers, the server cannot enforce flow control for that queue. Also, if a topic has inactive durable subscriptions or no current subscribers, the server cannot enforce flow control for that topic. For topics, if flow control is set on a specific topic (for example, `foo.bar`), then flow control is enforced as long as there are subscribers to that topic or any parent topic (for example, if there are subscribers to `foo.*`).

Multicast and Flow Control

If a multicast channel exceeds its maximum transmission rate, as determined by the `maxrate` of the channel definition in the `channels.conf` configuration file, the server may develop a backlog of messages. If `flow_control` parameter in the `tibemsd.conf` file is disabled, these messages are buffered in the server until they can be sent over multicast. The channel backlog can be determined using the `show channel` command in the administration tool, or through the `ChannelInfo` object in the administration API.

When the `flow_control` parameter is enabled, the EMS server checks for backlog before sending a response to the message producers publishing to multicast-enabled topics. If a message backlog exists, the server delays sending a response to the message producer until the backlog has been cleared. This causes the message producer to decrease the rate at which it sends messages to the topic.



The destination property `flowControl` is not used when determining whether flow control is to be engaged or disengaged by a multicast channel.

Routes and Flow Control

For global topics where messages are routed between servers, flow control can be specified for a topic on either the server where messages are produced or the server where messages are received. Flow control is not relevant for queue messages that are routed to another server.

If the `flowControl` property is set on the topic on the server receiving the messages, when the pending message size limit is reached, messages are not forwarded by way of the route until the topic subscriber receives enough messages to lower the pending message size below the specified limit.

If the `flowControl` property is set on the topic on the server sending the messages, the server may block any topic publishers when sending new messages if messages cannot be sent quickly enough by way of the route. This could be due to network latency between the routed servers or it could be because flow control on the other server is preventing new messages from being sent.

Destination Bridges and Flow Control

Flow control can be specified on bridged destinations. If you wish the flow of messages sent over the bridge to slow down when receivers on the bridged-to destination cannot process the messages quickly enough, you must set the `flowControl` property on both destinations on either side of the bridge.

Flow Control, Threads and Deadlock

When using flow control, you must be careful to avoid potential deadlock.

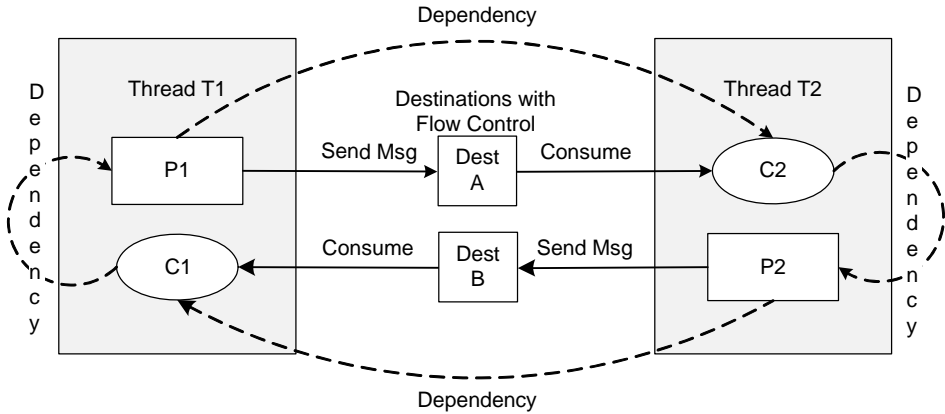
When flow control is in effect for a destination, producers to that destination can block waiting for flow control signals from the destination's consumers. If any of those consumers are within the same thread of program control, a potential for deadlock exists. Namely, the producer will not unblock until the destination contains fewer messages, and the consumer in the blocked thread cannot reduce the number of messages.

The simplest case to detect is when producer and consumer are in the same session (sessions are limited to a single thread). But more complex cases can arise. Deadlock can even occur across several threads, or even programs on different hosts, if dependencies link them. For example, consider the situation in [Figure 14](#):

- Producer P1 in thread T1 has a consumer C2 in thread T2.
- Producer P2 in T2 has a consumer C1 in T1.
- Because of the circular dependency, deadlock can occur if either producer blocks its thread waiting for flow control signals.

The dependency analysis is analogous to mutex deadlock. You must analyze your programs and distributed systems in a similar way to avoid potential deadlock.

Figure 14 Flow Control Deadlock across Two Threads



Delivery Delay



This feature is currently available only with EMS clients for Java.

The delivery delay feature allows the message producer to specify the earliest time at which a message should be delivered to consumers. This is done by using the `setDeliveryDelay()` method to set the minimum length of time that must elapse after a message is sent before the EMS server may deliver the message to a consumer.

Whenever a message is sent to destination *dest* with a non-zero delivery delay for the first time, the server dynamically creates a queue named `$sys.delayed.q.dest` when *dest* is a queue, or `$sys.delayed.t.dest` when *dest* is a topic.

`$sys.delayed` queues support browsing and purging but do not support other permissions such as receive or send. They inherit destination limits, security, and storage selection properties from *dest*. However, note that a `$sys.delayed.t` queue created for a topic that has the `secure` property cannot be browsed.

Note that the `$sys.delayed` queue corresponding to a destination takes any `maxmsgs` property setting from the destination. That is, if *dest* has property `maxmsgs` set to *X*, its `$sys.delayed` queue also has `maxmsgs` set to *X*. This doubles the number of messages that can potentially be held for *dest* in the server.

If the `maxmsgs` limit has been reached and the destination has the property `overflowPolicy=rejectIncoming`, when the delivery delay expires for a message one of two things can happen. If the message has the `JMS_TIBCO_PRESERVE_UNDELIVERED` set to `true`, it is put on the `$sys.undelivered` queue. Otherwise, the message is discarded.

Note that, when delivery delay is enabled for a topic, the behavior of `overflowPolicy=default` mimics that of a queue. That is, when `maxbytes` or `maxmsgs` has been reached, new messages are rejected by the server and an error is returned to the producer.

Chapter 4 **Getting Started**

This chapter provides a quick introduction to setting up a simple EMS configuration and running some sample client applications to publish and subscribe users to a topic.

Topics

- [About the Sample Clients, page 94](#)
- [Compiling the Sample Java Clients, page 95](#)
- [Creating Users with the EMS Administration Tool, page 96](#)
- [Point-to-Point Messaging Example, page 98](#)
- [Publish and Subscribe Messaging Example, page 99](#)
- [Multicast Messaging Example, page 104](#)

About the Sample Clients

The EMS sample clients were designed to allow you to run TIBCO Enterprise Message Service with minimum start-up time and coding.

The *EMS_HOME/samples* directory contains several subdirectories. The */emsc* subdirectory contains samples related to the Central Administration interface. The */c*, */cs*, and */java* subdirectories contain the C, .NET and Java sample clients.

In this chapter, you will compile and run the Java sample clients. For information on how to run the C, .NET, and Central Administration sample clients, see the readme files in their respective directories.

The *EMS_HOME/samples/java* directory contains three sets of files:

- Sample clients for TIBCO Enterprise Message Service implementation.
- The *JNDI* subdirectory contains sample clients that use the JNDI lookup technique.
- The *tibrv* subdirectory contains sample clients that demonstrate the interoperation of TIBCO Enterprise Message Service with TIBCO Rendezvous applications.

In this chapter, you will use some of the sample clients in the *EMS_HOME/samples/java* directory. For information on compiling and running the other sample clients, see the Readme files in their respective folders.

Compiling the Sample Java Clients

To compile and run the sample Java clients you need to execute "setup" script, which is located in the *EMS_HOME/samples/java* directory. On Windows systems, the setup file is *setup.bat*. On Unix systems, the setup file is *setup.sh*.

To compile the sample client files:

1. Make sure you have JDK 1.7 or greater installed and that you've added the bin directory to your PATH variable.
2. Open a command line or console window, and navigate to the *EMS_HOME/samples/java* directory.
3. Open the correct setup script file and verify that the *TIBEMS_ROOT* environment variable identifies the correct pathname to your *EMS_HOME* directory. For example, on a Windows system this might look like:

```
set TIBEMS_ROOT=C:\tibco\ems\8.0
```
4. Enter setup to set the environment and classpath:

```
> setup
```
5. Compile the samples:

```
> javac -d . *.java
```

This compiles all the samples in the directory, except for those samples in the *JNDI* and *tibrv* subdirectories.

If the files compile successfully, the class files will appear in the *EMS_HOME/samples/java* directory. If they do not compile correctly, an error message appears.

Creating Users with the EMS Administration Tool

This section describes how to start the EMS server and use the administration tool to create two new users.



All of the parameters you set using the administration tool in this chapter can also be set by editing the configuration files described in [Using the Configuration Files on page 185](#). You can also programmatically set parameters using the C, .NET, or Java APIs. Parameters set programmatically by a client are only set for the length of the session.

Start the EMS Server and EMS Administration Tool

In this example, you will create topics and users using the EMS administration tool. You must first start the EMS server before starting the EMS administration tool.

Start the EMS server

Start the EMS server as described in [Running the EMS Server on page 107](#).



On a computer running Windows, you can also start the EMS server from the Start menu, following the path **Programs > TIBCO > TIBCO EMS 8.0 > Start EMS server**.

Start the Administration Tool and Connect to the EMS Server

Start the EMS administration tool as described in [Starting the EMS Administration Tool on page 124](#).



On a computer running Windows, you can also start the administration tool from the Start menu, following the path **Programs > TIBCO > TIBCO EMS 8.0 > Start EMS administration tool**.

After starting the administration tool, connect it to the EMS server.

To connect the EMS administration tool to the EMS server, execute one of the following commands:

- If you are using TIBCO Enterprise Message Service on a single computer, type **connect** in the command line of the Administration tool:

```
> connect
```

You will be prompted for a login name. If this is the first time you've used the EMS administration tool, follow the procedure described in [When You First Start tibemsadmin on page 126](#).

Once you have logged in, the screen will display:

```
connected to tcp://localhost:7222
tcp://localhost:7222>
```

- If you are using TIBCO Enterprise Message Service in a network, use the connect server command as follows:

```
> connect [server URL] [user-name] [password]
```

For more information on this command, see [connect on page 130](#).

For further information on the administration tool, see [Starting the EMS Administration Tool on page 124](#) and [Command Listing on page 128](#).

Create Users

Once you have connected the administration tool to the server, use the create user command to create two users.

In the administration tool, enter:

```
tcp://localhost:7222> create user user1
tcp://localhost:7222> create user user2
```

The tool will display messages confirming that user1 and user2 have been created.

You have now created two users. You can confirm this with the [show users](#) command:

```
tcp://localhost:7222> show users
User Name      Description
user1
user2
```

For more information on the create user command, refer to [create user on page 133](#).

Point-to-Point Messaging Example

This section demonstrates how to use point-to-point messaging, as described in [Point-to-Point on page 3](#).

Create a Queue

In the point-to-point messaging model, client send messages to and receive messages from a queue.

To create a new queue in the administration tool, use the `create queue` command to create a new queue named `myQueue`:

```
tcp://localhost:7222> create queue myQueue
```

For more information on the `create queue` command, refer to [create queue on page 132](#). For more information on the `commit` command, see [commit on page 129](#) and [autocommit on page 129](#).

Start the Sender and Receiver Clients

1. Open two command line windows and in each window navigate to the `EMS_HOME/samples/java` folder.
2. In each command line window, enter `setup` to set the environment and classpath:

```
> setup
```
3. In the first command line window, execute the `tibjmsMsgProducer` application to direct `user1` to place some messages to the `myQueue` queue:

```
> java tibjmsMsgProducer -queue myQueue -user user1 Hello User2
```
4. In the second command line window, execute the `tibjmsMsgConsumer` client to direct `user2` to read the messages from the message queue:

```
> java tibjmsMsgConsumer -queue myQueue -user user2
```

The messages placed on the queue are displayed in the receiver's window.



Messages placed on a queue by the sender are persistent until read by the receiver, so you can start the sender and receiver clients in any order.

Publish and Subscribe Messaging Example

In this section, you will execute a message producer client and two message consumer clients that demonstrate the publish/subscribe messaging model described in [Publish and Subscribe on page 4](#). This example is not intended to be comprehensive or representative of a robust application.

To execute the client samples, you must give them commands from within the sample directory that contains the compiled samples. For this exercise, open three separate command line windows and navigate to the *EMS_HOME/samples/java* directory in each window.

For more information on the samples, refer to the *readme* within the sample directory. For more information on compiling the samples, refer to [Compiling the Sample Java Clients on page 95](#).

Create a Topic

In the publish/subscribe model, you publish and subscribe to topics.

To create a new topic in the administration tool, use the `create topic` command to create a new topic named `myTopic`:

```
tcp://localhost:7222> create topic myTopic
```

For more information on the `create topic` command, refer to [create topic on page 133](#). For more information on the `commit` command, see [commit on page 129](#) and [autocommit on page 129](#).

Start the Subscriber Clients

You start the subscribers first because they enable you to observe the messages being received when you start the publisher.

To start `user1` as a subscriber:

1. In the first command line window, navigate to *EMS_HOME/samples/java*.
2. Enter `setup` to set the environment and classpath:

```
> setup
```
3. Execute the `tibjmsMsgConsumer` client to assign `user1` as a subscriber to the `myTopic` topic:

```
> java tibjmsMsgConsumer -topic myTopic -user user1
```

The screen will display a message showing that `user1` is subscribed to `myTopic`.

To start user2 as a subscriber:

1. In the second command line window, navigate to the `EMS_HOME/samples/java` folder.
2. Enter `setup` to set the environment and classpath:
`> setup`
3. Execute the `tibjmsMsgConsumer` application to assign user2 as a subscriber to the `myTopic` topic:
`> java tibjmsMsgConsumer -topic myTopic -user user2`

The screen will display a message showing that user2 is subscribed to `myTopic`.



The command windows do not return to the prompt when the subscribers are running.

Start the Publisher Client and Send Messages

Setting up the publisher is very similar to setting up the subscriber. However, while the subscriber requires the name of the topic and the user, the publisher also requires messages.

To start the publisher:

1. In the third command line window, navigate to the `EMS_HOME/samples/java` folder.
2. Enter `setup` to set the environment and classpath:
`> setup`
3. Execute the `tibjmsMsgProducer` client to direct user1 to publish some messages to the `myTopic` topic:
`> java tibjmsMsgProducer -topic myTopic -user user1 hello user2`
 where 'hello' and 'user2' are separate messages.



In this example, user1 is both a publisher and subscriber.

The command line window will display a message stating that both messages have been published:

```
Publishing on topic 'myTopic'
Published message: hello
Published message: user2
```

After the messages are published, the command window for the publisher returns to the prompt for further message publishing.



Note that if you attempt to use the form:

```
java tibjmsMsgProducer -topic myTopic -user user1
```

without adding the messages, you will see an error message, reminding you that you must have at least one message text.

The first and second command line windows containing the subscribers will show that each subscriber received the two messages:

```
Subscribing to topic: myTopic
Received message: TextMessage={ Header={
JMSMessageID={ID:EMS-SERVER.97C44203CDF A:1}
JMSDestination={Topic[myTopic]} JMSReplyTo={null}
JMSDeliveryMode={PERSISTENT} JMSRedelivered={false}
JMSCorrelationID={null} JMSType={null} JMSTimestamp={Tue Mar 21
12:04:56 PST 2006} JMSEExpiration={0} JMSPriority={4} }
Properties={} Text={hello} }
Received message: TextMessage={ Header={
JMSMessageID={ID:EMS-SERVER.97C44203CDFA:2}
JMSDestination={Topic[myTopic]} JMSReplyTo={null}
JMSDeliveryMode={PERSISTENT} JMSRedelivered={false}
JMSCorrelationID={null} JMSType={null} JMSTimestamp={Tue Mar 21
12:04:56 PST 2006} JMSEExpiration={0} JMSPriority={4} }
Properties={} Text={user2} }
```

Create a Secure Topic

In this example, you make `myTopic` into a secure topic and grant `user1` permission to publish to the `myTopic` and `user2` permission to subscribe to `myTopic`.

Add the secure Property to the Topic

When the secure property is added to a topic, only users who have been assigned a certain permission can perform the actions allowed by that permission. For example, only users with publish permission on the topic can publish, while other users cannot publish.

If the secure property is not added to a topic, all authenticated users have all permissions (publish, subscribe, create durable subscribers) on that topic.

For more information on the secure property, see the section about [secure](#), [page 71](#). For more information on topic permissions, see [Chapter 8, Authentication and Permissions](#), on [page 265](#).

To enable server authorization and add the [secure](#) property to a topic, do the following steps:

1. In each subscriber window, enter Control-C to stop each subscriber.
2. In the administration tool, use the `set server` command to enable the [authorization](#) property:

```
tcp://localhost:7222> set server authorization=enabled
```

The `authorization` property enables checking of permissions set on destinations.
3. Enter the following command to add the `secure` property to a topic named `myTopic`:

```
tcp://localhost:7222> addprop topic myTopic secure
```

For more information on the `set server` command, refer to [set server on page 143](#). For more information on the `addprop topic` command, refer to [addprop topic on page 129](#).

Grant Topic Access Permissions to Users

To see how permissions affect the ability to publish and receive messages, grant publish permission to `user1` and subscribe permission to the `user2`.

Use the `grant topic` command to grant permissions to users on the topic `myTopic`.

In the administration tool, enter:

```
tcp://localhost:7222> grant topic myTopic user1 publish
tcp://localhost:7222> grant topic myTopic user2 subscribe
```

For more information on the `grant topic` command, refer to [grant topic on page 137](#).

Start the Subscriber and Publisher Clients

Start the subscribers, as described in [Start the Subscriber Clients on page 99](#). Note that you cannot start `user1` as a subscriber because `user1` has permission to publish, but not to subscribe. As a result, you receive an exception message including the statement:

```
Operation not permitted.
```

`User2` should start as a subscriber in the same manner as before.

You can now start `user1` as the publisher and send messages to `user2`, as described in [Start the Publisher Client and Send Messages on page 100](#).

Create a Durable Subscriber

As described in [Publish and Subscribe on page 4](#), subscribers, by default, only receive messages when they are active. If messages are published when the subscriber is not available, the subscriber does not receive those messages. You can create durable subscriptions, where subscriptions are stored on the server and subscribers can receive messages even if it was inactive when the message was originally delivered.

In this example, you create a durable subscriber that stores messages published to topic `myTopic` on the EMS server.

To start `user2` as a durable subscriber:

1. In the a command line window, navigate to the `EMS_HOME/samples/java` folder.

2. Enter `setup` to set the environment and classpath:

```
> setup
```

3. Execute the `tibjmsDurable` application to assign `user2` as a durable subscriber to the `myTopic` topic:

```
> java tibjmsDurable -topic myTopic -user user2
```

4. In the administration tool, use the `show durables` command to confirm that `user2` is a durable subscriber to `myTopic`:

```
tcp://localhost:7222> show durables
  Topic Name      Durable      User      Msgs      Size
* myTopic        subscriber    user2      0         0.0 Kb
```

5. In the subscriber window, enter `Ctrl+C` to stop the subscriber.
6. In another command line window, execute the `tibjmsMsgProducer` client, as described in [Start the Publisher Client and Send Messages on page 100](#):

```
> java tibjmsMsgProducer -topic myTopic -user user1 hello user2
```

7. Restart the subscriber:

```
> java tibjmsDurable -topic myTopic -user user2
```

The stored messages are displayed in the subscriber window.

8. Enter `Ctrl+C` to stop the subscriber and then unsubscribe the durable subscription:

```
> java tibjmsDurable -unsubscribe
```

The subscriber is no longer durable and any additional messages published to the `myTopic` topic are lost.

Multicast Messaging Example

This section demonstrates how to use multicast messaging, as described in [Multicast on page 6](#).

In this example, you will enable multicast in the EMS server and configure a multicast channel, over which the server can broadcast multicast messages. You will also create a multicast-enabled topic named `multicastTopic` and associate it with the multicast channel, allowing subscribers to receive messages published to `multicastTopic` over multicast.

Multicast channels can only be configured statically by modifying the configuration files. There are no commands in the administration tool to configure multicast channels.

Stop the EMS Server

Stop the server by using the shutdown command in the administration tool:

```
tcp://localhost:7222> shutdown
```

You will be asked to restart the server once it has been configured for multicast.

Enable the EMS Server for Multicast

To enable multicast in the server, set the `multicast` property to `enabled` in the `tibemsd.conf` configuration file:

```
multicast = enabled
```

Create a Multicast Channel

The EMS server broadcasts messages to consumers over multicast channels. Each channel has a defined multicast address and port. Messages published to a multicast-enabled topic are sent by the server and received by the subscribers on these multicast channels.

To create a multicast channel, add the following definition to the multicast channels configuration file, `channels.conf`:

```
[multicast-1]  
address=234.5.6.7:1
```

Start the EMS Server

Start the EMS server as described in [Running the EMS Server on page 107](#).



On a computer running Windows, you can also start the EMS server from the Start menu, following the path **Programs > TIBCO > TIBCO EMS 8.0 > Start EMS server**.

In the administration tool, use the `show topics` command to confirm that `multicastTopic` is multicast-enabled as indicated by a '+' in the M column:

```
tcp://localhost:7222> show topics
  Topic Name      SNFGEIBCTM  Subs   Durs   Msgs   Size
  multicastTopic  -----+   0      0      0     0.0 Kb
```

Enable a Topic for Multicast

In order to make a topic multicast-enabled it must be associated with a multicast channel through its `channel` property.

To create a multicast-enabled topic, use the administration tool to issue the following command:

```
> create topic multicastTopic channel=multicast-1
```

Start the Multicast Daemon

Start the Multicast Daemon as described in [Starting the Multicast Daemon on page 379](#).

Start the Subscriber Client

Creating a multicast subscriber follows the same steps as creating a non-multicast subscriber, except that a multicast subscriber requires a session acknowledgment mode of `com.tibco.tibjms.Tibjms.NO_ACKNOWLEDGE`.

To start `user1` as a multicast subscriber:

1. In a command line window, navigate to the `EMS_HOME/samples/java` folder.
2. Enter `setup` to set the environment and classpath:


```
> setup
```

3. Execute the `tibjmsMsgConsumer` client to assign `user1` as a subscriber to the `multicastTopic` topic with a Session acknowledgment mode of `NO_ACKNOWLEDGE`:

```
> java tibjmsMsgConsumer -topic multicastTopic -user user1 -ackmode NO
```

4. In the administration tool, use the `show consumers` command to confirm that `user1` is a multicast subscriber to `multicastTopic` as indicated by a `+` in the `M` column:

```
tcp://optimist:7222> show consumers topic=multicastTopic
```

Id	Conn	User	T	Topic	SASNM	Pend Msgs	Pend Size	Uptime
2	4	user1	T	multicastTopic	+N--+	0	0	0:03:17

Start the Publisher Client and Send Messages

Setting up a client to publish multicast message is no different from setting up a client to send publish and subscribe messages. Because the topic is enabled for multicast in the EMS server, the message producer does not need to follow any additional steps.

To create the message publisher:

1. In a new command line window, navigate to the `EMS_HOME/samples/java` folder.
2. Enter `setup` to set the environment and classpath:

```
> setup
```

3. Execute the `tibjmsMsgProducer` client to direct `user1` to publish some messages to the `multicastTopic` topic:

```
> java tibjmsMsgProducer -topic multicastTopic -user user1 hello  
multicast
```

where `'hello'` and `'multicast'` are separate messages.



In this example, `user1` is both a publisher and subscriber.

The messages are displayed in the subscriber's window.

Chapter 5

Running the EMS Server

To use TIBCO Enterprise Message Service with your applications, the TIBCO Enterprise Message Service Server must be running. The server and the clients work together to implement TIBCO Enterprise Message Service. The server implements all types of message persistence and no messages are stored on the client side.

Topics

- [Starting and Stopping the EMS Server, page 108](#)
- [Running the EMS Server as a Windows Service, page 112](#)
- [Error Recovery Policy, page 115](#)
- [Security Considerations, page 116](#)
- [How EMS Manages Access to Shared Store Files, page 120](#)

Starting and Stopping the EMS Server

This section describes how to start and stop the EMS server.



On a computer running Microsoft Windows, you can start the EMS server from the Start menu, following the path: **Programs > TIBCO > TIBCO EMS 8.0 > Start EMS Server**



On Windows systems the `tibemsd.exe` and `tibemsmcd.exe` files run as administrator to enable multicast functionality and to let the `tibemsd.exe` modify the configuration files.

Starting the EMS Server Using the Default Configuration

To start the EMS server from the command line using the preconfigured setup, navigate to `EMS_HOME/bin` and run the script:

On UNIX `tibemsd.sh`

On Windows `tibemsd.bat`

Running the script starts the EMS server using the configuration files in the default location, `config-file-directory\cfmgmt\ems\data` directory, where the `config-file-directory` corresponds to the Configuration Directory specified during installation.

Starting the EMS Server Using JSON Configuration



Users using the Central Administration feature must start the EMS server in JSON mode. This is done from the command line, using the `-config` option to specify the JSON configuration file. For more information, see *JSON Configuration Files* in the *TIBCO Enterprise Message Service Central Administration* guide.

Your JSON-configured `tibemsd` must be running before it can be added to the Central Administration server list. To start the TIBCO Enterprise Message Service server using the JSON configuration file:

1. From the command line, navigate to `EMS_HOME/bin`.
2. Enter the following command and option:

```
tibemsd -config json-file-path
```

where `json-file-path` is the path to your JSON configuration file. For example:

```
tibemsd -config /tibemsconfig/tibemsd.json
```

When started using the JSON configuration, the `tibemsd` silently ignores any unknown parameters. For example, no configuration errors are thrown if the `tibemsd.json` file contains an obsolete parameter.



For information on converting `.conf` configuration files to JSON configuration files, see [Converting Server Configuration Files to JSON](#) in the *TIBCO Enterprise Message Service Central Administration* guide.

Starting Fault Tolerant Server Pairs

In Central Administration, fault tolerant pairs share a single JSON configuration file. Primary and secondary server roles are determined when the servers are started.

Start the primary EMS server as usual. Start the secondary server using the `-secondary` flag. For example, where the JSON configuration file is `tibemsd.json`:

To start the primary server: `tibemsd -config tibemsd.json`

To start the secondary server: `tibemsd -config tibemsd.json -secondary`

For more information, see [Configuring Fault Tolerance in Central Administration on page 502](#).

Starting the EMS Server Using Options

To start the EMS server from the command line using options:

Task A Navigate to the data subdirectory.

The preconfigured EMS server files are located in the `config-file-directory\cfmgmt\ems\data` directory, where the *config-file-directory* corresponds to the Configuration Directory specified during installation. For more information, see *Installing TIBCO Enterprise Message Service in TIBCO Enterprise Message Service Installation*.

Change the directory to the installed data directory:

On UNIX `./tibemsd -config "config-file-directory/cfmgmt/ems/data/tibemsd.conf"`

On Windows `.\tibemsd -config "config-file-directory\cfmgmt\ems\data\tibemsd.conf"`

Alternately, change to the `EMS_HOME\samples\config` directory and create a datastore directory (or other directories as needed) to use the sample configuration files there.

The EMS server dynamically loads the SSL and compression shared libraries, rather than statically linking them. If the `tibemspd` executable is executed from the data directory, it automatically locates these libraries. If the server is moved elsewhere, the shared library directory must be moved as well.

Task B Start the tibemspd

Type `tibemspd [options]`

where *options* are described in [Table 14](#). The command options to `tibemspd` are similar to the parameters you specify in `tibemspd.conf`, and the command options override any value specified in the parameters. See [tibemspd.conf on page 187](#) for more information about configuration parameters.

Table 14 tibemspd Options

Option	Description
<code>-config config file name</code>	<p><i>config file name</i> is the name of the main configuration file for <code>tibemspd</code> server. Default is <code>tibemspd.conf</code>.</p> <p>For example, to start an EMS server using the default JSON configuration file, use:</p> <pre>tibemspd -config tibemspd.json</pre>
<code>-trace items</code>	<p>Specifies the trace items. These items are not stored in the configuration file. The value has the same format as the value of <code>log_trace</code> parameter specified with <code>set server</code> command of the administration tool; see Tracing on the Server on page 447.</p>
<code>-secondary</code>	<p>Specifies the secondary server in a fault tolerant pair. This option is only valid for EMS servers started using JSON config</p>
<code>-ssl_password string</code>	<p>Private key password.</p>
<code>-ssl_trace</code>	<p>Print the certificates loaded by the server and do more detailed tracing of SSL-related situation.</p>
<code>-ssl_debug_trace</code>	<p>Turns on tracing of SSL connections.</p>
<code>-ft_active active_url</code>	<p>URL of the active server. If this server can connect to the active server, it will act as a backup server. If this server cannot connect to the active server, it will become the active server.</p>

Table 14 *tibemsd Options (Cont'd)*

Option	Description
<code>-ft_heartbeat</code> <i>seconds</i>	Heartbeat signal for the active server, in seconds. Default is 3.
<code>-ft_activation</code> <i>seconds</i>	Activation interval (maximum length of time between heartbeat signals) which indicates that active server has failed. Set in seconds: default is 10. This interval should be set to at least twice the heartbeat interval.
<code>-forceStart</code>	Causes the sever to delete corrupted messages in the store files, allowing the server to start even if it encounters errors. Note that using this option causes data loss, and it is important to backup store files before using <code>-forceStart</code> . See Error Recovery Policy on page 115 for more information.

Stopping the EMS Server

You can stop the EMS server by means of the [shutdown](#) command from the EMS Administration Tool.

Running the EMS Server as a Windows Service

Some situations require the EMS server and multicast daemon processes to start automatically. You can satisfy this requirement by registering these with the Windows service manager. The `emsntsrg` utility facilitates registry.

emsntsrg

The `emsntsrg` utility registers or unregisters the EMS server daemon or the EMS multicast daemon as a Windows service.



This utility applies only to Microsoft **Windows** (all supported versions, including 2003, XP, and Vista).

Syntax `emsntsrg /i [/a] service_name emsntsct_directory service_directory [arguments] [suffix]`
`emsntsrg /r [service_name] [suffix]`

Remarks Some situations require the EMS server processes to start automatically. You can satisfy this requirement by registering these with the Windows service manager. This utility facilitates registry.

Restrictions You must have administrator privileges to change the Windows registry.

Location Locate this utility program as an executable file in the EMS `bin` directory.

Parameter	Description
/i	Insert a new service in the registry (that is, register a new service).
/a	Automatically start the new service. Optional with /i.
/?	Display usage.
service_name	Insert or remove a service with this base name. When inserting a service, this parameter is required, and must be <code>tibemsd</code> or <code>tibemsmcd</code> . When removing a service, this parameter is optional. However, if it is present, it must be <code>tibemsd</code> or <code>tibemsmcd</code> .

Parameter	Description
<i>emsntsct_directory</i>	<p>Use this directory pathname to specify the location of the <code>emsntsct.exe</code> executable. The <code>emsntsrg</code> utility registers the <code>emsntsct.exe</code> program as a windows service. The <code>emsntsct.exe</code> program then invokes the associated <code>tibemsd</code> or <code>tibemsmcd</code>.</p> <p>By default, <code>emsntsct.exe</code> is located in <code>EMS_HOME\bin</code>.</p> <p>This parameter is only required when installing a service.</p>
<i>service_directory</i>	Use this directory pathname to locate the service executable, either <code>tibemsd</code> or <code>tibemsmcd</code> . Required.
<i>arguments</i>	<p>Supply command line arguments. Optional with <code>/i</code>.</p> <p>Enclose the entire arguments string in double quote characters.</p>
<i>suffix</i>	When registering more than one instance of a service, you can use this suffix to distinguish between them in the Windows services applet. Optional.
<code>/r</code>	Remove a service from the registry.

Register To register `tibemsd` as a Windows service, run the utility with this command line:

```
emsntsrg /i [/a] tibemsd emsntsct_directory tibemsd_directory [arguments]
[suffix]
```

To register `tibemsmcd` as a Windows service, run the utility with this command line:

```
emsntsrg /i [/a] tibemsmcd emsntsct_directory tibemsmcd_directory [arguments]
[suffix]
```

Example 1 This simple example registers one `tibemsd` service:

```
emsntsrg /i tibemsd C:\tibco\ems\8.0\bin C:\tibco\ems\8.0\bin
```

Example 2 This example registers a service with command line arguments:

```
emsntsrg /i tibemsd C:\tibco\ems\8.0\bin C:\tibco\ems\8.0\bin
"-trace DEFAULT"
```

Example 3 This pair of example commands registers two `tibemsd` services with different configuration files. In this example, the numerical suffix and the configuration directory both reflect the port number that the service uses.

```
emsntsrg /i tibemsd C:\tibco\ems\8.0\bin C:\tibco\ems\8.0\bin
"-config C:\tibco\ems\8.0\7222\tibemsd.conf" 7222
```

```
emsntsrg /i tibemsd C:\tibco\ems\8.0\bin C:\tibco\ems\8.0\bin
"-config C:\tibco\ems\8.0\7223\tibemsd.conf" 7223
```

Notice these aspects of this example:

- When installing `tibemsd`, if you supply a `-config` argument, the service process finds the directory containing the main configuration file (`tibemsd.conf`), and creates all secondary configuration files in that directory. In this example, each service uses a different configuration directory.
- When you register several EMS services, you must avoid configuration conflicts. For example, two instances of `tibemsd` cannot listen on the same port.

Example 4 This example registers one multicast daemon service:

```
emsntsrg /i tibemsmcd C:\tibco\ems\8.0\bin C:\tibco\ems\8.0\bin
```

Example 5 This example registers a multicast daemon service with command line arguments:

```
emsntsrg /i tibemsmcd C:\tibco\ems\8.0\bin C:\tibco\ems\8.0\bin
"-logfile c:\tibemsmcd.log"
```

Note that specifying a log file can help identify conflicts that might prevent the multicast daemon service from starting.

Example 6 This pair of example commands registers two multicast daemon services with different ports. In this example, the numerical suffix reflects the port number that the service uses.

```
emsntsrg /i tibemsmcd C:\tibco\ems\8.0\bin C:\tibco\ems\8.0\bin
"-listen 7345" 7345
```

```
emsntsrg /i tibemsmcd C:\tibco\ems\8.0\bin C:\tibco\ems\8.0\bin
"-listen 7346" 7346
```

Remove To unregister a service, run the utility with this command line:

```
emsntsrg /r [service_name] [suffix]
```

Both parameters are optional. If the *service_name* is present, it must be `tibemsd` or `tibemsmcd`. To supply the *suffix* parameter, you must also supply the *service_name*. When both parameters are absent, the utility removes the services named `tibemsd` and `tibemsmcd`.

Command Summary To view a command line summary, run the utility with this command line:

```
emsntsrg
```

Windows Services Applet The Windows services applet displays the name of each registered service. For EMS services, it also displays this additional information:

- The suffix (if you supply one)
- The process ID (PID)—when the service is running

Error Recovery Policy

During startup the EMS server can encounter a number of errors while it recovers information from the store files. Potential errors include:

- Low-level file errors. For example, corrupted disk records.
- Low-level object-specific errors. For example, a record that is missing an expected field.
- Inter-object errors. For example, a session record with no corresponding connection record.

When the EMS server encounters one of these errors during startup, the recovery policy is:

- By default, the server exits startup completely when a corrupt disk record error is detected. Because the state can not be safely restored, the server can not proceed with the rest of the recovery. You can then examine your configuration settings for errors. If necessary, you can then copy the store and configuration files for examination by TIBCO Support.
- You can direct the server to delete bad records by including the `-forceStart` command line option. This prevents corruption of the server runtime state.
- The server exits if it runs out of memory during startup.

It is important to backup the store files before restarting the server with the `-forceStart` option, because data will be lost when the problematic records are deleted.

Keep in mind that different type of records are stored in the store files. The most obvious are the persistent JMS Messages that your applications have sent. However, other internal records are also stored. If a consumer record used to persist durable subscriber state information were to be corrupted and later deleted with the `-forceStart` option, all JMS messages that were persisted (and valid in the sense that they were not corrupted) would also be lost because the durable subscription itself would not be recovered.

When running in this mode, the server still reports any errors found during the recovery, but problematic records are deleted and the recovery proceeds. This mode may report more issues than are reported without the `-forceStart` option, because without that flag the server stops with the very first error.



We strongly recommended that you make a backup of all store files before restarting the server with the `-forceStart` option. The backup is useful when doing a postmortem analysis to find out what records were deleted with the `-forceStart` option.

Security Considerations



This section highlights information relevant to secure deployment. We recommend that all administrators read this section.

Secure Environment

To ensure secure deployment, EMS administration must meet the following criteria:

- **Correct Installation** EMS is correctly installed and configured.
- **Physical Controls** The computers where EMS is installed are located in areas where physical entry is controlled to prevent unauthorized access. Only authorized administrators have access, and they cooperate in a benign environment.
- **Domain Control** The operating system, file system and network protocols ensure domain separation for EMS, to prevent unauthorized access to the server, its configuration files, LDAP servers, etc.
- **Benign Environment** Only authorized administrators have physical access or domain access, and those administrators cooperate in a benign environment.

Destination Security

Three interacting factors affect the security of destinations (that is, topics and queues). In a secure deployment, you must properly configure all three of these items:

- The server's authorization parameter (see [Authorization Parameter](#), below)
- The secure property of individual destinations (see [secure on page 71](#))
- The ACL permissions that apply to individual destinations (see [Authentication and Permissions on page 265](#))

Authorization Parameter

The server's [authorization](#) parameter acts as a master switch for checking permissions for connection requests and operations on secure destinations. The default value of this parameter is `disabled`—the server does not check any permissions, and allows all operations. For secure deployment, you must enable this parameter.

Admin Password

For ease in installation and initial testing, the default setting for the `admin` password is no password at all. Until you set an actual password, the user `admin` can connect without a password. Once the administrator password has been set, the server always requires it.

To configure a secure deployment, the administrator must change the `admin` password immediately after installation; see [Assign a Password to the Administrator on page 126](#).

Connection Security

When `authorization` is enabled, the server requires a name and password before users can connect. Only authenticated users can connect to the server. The form of authentication can be either an X.509 certificate or a username and password (or both).

When `authorization` is disabled, the server does not check user authentication; all user connections are allowed. However, even when `authorization` is disabled, the user `admin` must still supply the correct password to connect to the server.

Even when `authorization` is enabled, the administrator (`admin`) may explicitly allow anonymous user connections, which do not require password authorization. To allow these connections, create a user with the name `anonymous` and no password.



Creating the user `anonymous` does not mean that `anonymous` has all permissions. Individual topics and queues can still be secure, and the ability to use these destinations (either sending or receiving) is controlled by the access control list of permissions for those destinations. The user `anonymous` can access only non-secure destinations.

Nonetheless, this feature (anonymous user connections) is outside the tested configuration of EMS security certification.

For more information on destination security, refer to the destination property [secure on page 71](#), and [Create Users on page 97](#).

Communication Security

For communication security between servers and clients, and between servers and other servers, you must explicitly configure SSL within EMS; see [Using the SSL Protocol on page 465](#).

SSL communication requires software to implement SSL on both server and client. The EMS server includes the OpenSSL implementation. Java client programs must use either JSSE (part of the Java environment) or separately purchased SSL software from Entrust; neither of these are part of the EMS product. C client programs can use the OpenSSL library shipped with EMS.

Sources of Authentication Data

The server uses only one source of X.509 certificate authentication data, namely, the server parameter `ssl_server_trusted` (its value is set in EMS an configuration file). See [ssl_server_trusted on page 227](#).

The server can use three sources of secure password authentication data:

- Local data from the EMS configuration files.
- External data from an LDAP.
- A user-supplied JAAS LoginModule.

You must safeguard the security of EMS configuration files and LDAP servers.

Timestamp

The administration tool can either include or omit a timestamp associated with the output of each command. To ensure a secure deployment, you must explicitly enable the timestamp feature. Use the following administration tool command:

```
time on
```


Passwords



Passwords are a significant point of vulnerability for any enterprise. We recommend enforcing strong standards for passwords.

For security equivalent to single DES (an industry minimum), security experts recommend passwords that contain 8–14 characters, with at least one upper case character, at least one numeric character, and at least one punctuation character.

EMS software does not automatically enforce such standards for passwords. You must enforce such policies within your organization.

Audit Trace Logs

Audit information is output to log files (and `stderr`), and is configured by the server parameters `log_trace` and `console_trace` (see [Tracing and Log File Parameters on page 220](#)).

The `DEFAULT` setting includes `+ADMIN`, so all administrative operations produce audit output. For further details, see [Table 72, Server Tracing Options, on page 448](#).

Audit information in log files is always timestamped.

Administrators can read and print the log files for audit review using tools (such as text editors) commonly available within all IT environments. EMS software does not include a special tool for audit review.

How EMS Manages Access to Shared Store Files

To prevent two EMS servers from using the same store file, each server restricts access to its store file for the duration of the server process. This section describes how EMS manages locked store files.

Windows On Windows platforms, servers use the standard Windows `CreateFile` function, supplying `FILE_SHARE_READ` as the `dwShareMode` (third parameter position) to restrict access to other servers.

UNIX On UNIX platforms, servers use the standard `fcntl` operating system call to implement cooperative file locking:

```
struct flock fl;  
int err;  
  
fl.l_type = F_WRLCK;  
fl.l_whence = 0;  
fl.l_start = 0;  
fl.l_len = 0;  
  
err = fcntl(file, F_SETLK, &fl);
```

To ensure correct locking, we recommend checking the operating system documentation for this call, since UNIX variants differ in their implementations.

Performance Tuning

By default, the TIBCO Enterprise Message Service server has the following general thread architecture:

- A single thread to process network traffic.
- One thread for each store.
- Additional threads for various background tasks such as expiring messages, connecting routes, and so on.

Setting Thread Affinity for Increased Throughput

If the default behavior of the EMS server cannot provide the required throughput and the EMS server machine has multiple cores, you can assign specific cores to the EMS threads that handle network traffic and stores.

For instance, with a 4-core machine, you can use the `processor_ids` parameter to assign core 0 and core 1 to handle network traffic. You can also use the store configuration `processor_id` parameter to assign core 2 to handle the `$sys.failsafe` store. This configuration causes the EMS server to create two threads that handle network traffic, and sets the affinity of them to core 0 and core 1 respectively. It also sets the affinity of the thread handling the store `$sys.failsafe` to core 2. No affinity is set for other threads.

Determining Core Allocation

The phrase "less is more" summarizes the best practices for EMS performance tuning.

1. If the default behavior provides sufficient throughput, do not change it.

When the EMS server does not set thread affinity, the operating system can better schedule EMS server threads to react to changing workloads on the machine. Also examine if the application is making efficient use of the API before changing the default behavior. For example, when performing persistent messaging operations, consider using multiple threads in the applications (each with its own session) or consider using local transactions to batch sends and acknowledgements.

2. Use the minimum number of cores to handle network traffic. Binding a single core may yield sufficient performance improvements over the default behavior, so start testing affinity there. Using excessive numbers of cores leads to greater thread contention for global data structures, which can reduce throughput and waste machine resources. Excessive numbers can also lead to

more unbalanced connection assignments. TIBCO tests have shown that three (or four under some workloads) is the maximum useful number for network traffic.

3. When setting core assignment for network traffic for persistent messaging, also set core assignment for stores in order to prevent contention between threads handling those tasks.

Network I/O Connections

When a client connects to the EMS server, the EMS server assigns it to one of the threads handling network traffic based on which of those threads have the fewest existing connections. This balances the total number of connections evenly across those threads.

Note that if all the connections to one thread are closed, the EMS server does not move existing connections from other threads in order to rebalance them.

Also note that the EMS server does not account for the traffic generated by those connections. For instance, the EMS server could assign ten connections to one thread and ten connections to another thread but still have an unbalanced state if the first ten connections account for 90% of all network traffic to the EMS server.

Other Considerations

- When assigning cores for EMS use, ensure that the Operating System does not schedule those cores for other processes.
- Assign cores on the same die if possible. This reduces cache sharing between dies. High levels of cache sharing between dies reduces memory performance.
- Hyper-threads are not real cores. Disable hyper-threading if possible. Do not assign cores to the EMS server such that it sets affinity for two "cores" that are actually sharing the same physical core by hyper-threading.

Chapter 6

Using the EMS Administration Tool

This chapter gives an overview of commands and use in the administration tool for TIBCO Enterprise Message Service.

Topics

- [Starting the EMS Administration Tool, page 124](#)
- [Naming Conventions, page 127](#)
- [Command Listing, page 128](#)

Starting the EMS Administration Tool

The EMS Administration Tool is located in your `EMS_HOME/bin` directory and is a stand-alone executable named `tibemsadmin` on UNIX and `tibemsadmin.exe` on Windows platforms.

On a computer running Windows, you can also start the administration tool from the Start menu, following the path **Programs > TIBCO > TIBCO EMS 8.0 > Start EMS administration tool**.

The EMS server must be started as described in [Chapter 5, Running the EMS Server, on page 107](#) before you start the EMS Administration Tool.



When a system uses shared configuration files, then actions performed using the administration tool take effect only when connected to the active server. If you have configured fault-tolerant servers and connect to the standby server, the administration tool will print a message notifying you of this. Additionally, if the administration tool is connected to the backup server, it will be disconnected when a switchover occurs.

Type `tibemsadmin -help` to display information about `tibemsadmin` startup parameters. All `tibemsadmin` parameters are optional.

[Table 15](#) lists options for `tibemsadmin`.

Table 15 *tibemsadmin Options*

Option	Description
<code>-help</code> or <code>-h</code>	Print the help screen.
<code>-script script-file</code>	Execute the specified text file containing <code>tibemsadmin</code> commands then quit. Any valid <code>tibemsadmin</code> command described in this chapter can be executed. Line breaks within the file delimit each command. That is, every command must be contained on a single line (no line breaks within the command), and each command is separated by a line break.
<code>-server server-url</code>	Connect to specified server.
<code>-user user-name</code>	Use this user name to connect to server.
<code>-password password</code>	Use this password to connect to server.

Table 15 *tibemsadmin* Options

Option	Description
<code>-ignore</code>	Ignore errors when executing script file. This parameter only ignores errors in command execution but not syntax errors in the script.
<code>-mangle [password]</code>	<p>Mangle the password and quit. Mangled string in the output can be set as a value of one of these passwords from the configuration files:</p> <ul style="list-style-type: none"> • server password • server SSL password • LDAP admin password • database password <p>If the password is not entered it is prompted for.</p>
<code>-ssl_trusted filename</code>	File containing trusted certificate(s). This parameter may be entered more than once if required.
<code>-ssl_identity filename</code>	File containing client certificate and (optionally) extra issuer certificate(s), and the private key.
<code>-ssl_issuer filename</code>	File containing extra issuer certificate(s) for client-side identity.
<code>-ssl_password password</code>	Private key or PKCS#12 password. If the password is required, but has not been specified, it will be prompted for.
<code>-ssl_noverifyhostname</code>	Do not verify hostname against the name on the certificate.
<code>-ssl_hostname name</code>	Name expected in the certificate sent by the host.
<code>-ssl_trace</code>	Show loaded certificates and certificates sent by the host.
<code>-ssl_debug_trace</code>	Show additional tracing, which is useful for debugging.



When a command specifies `-user` and `-password`, that information is not stored for later use. It is only used to connect to the server specified in the same command line. The user name and password entered on one command line are not reused with subsequent connect commands entered in the script file or interactively.

Examples

```
tibemsadmin -server "tcp://host:7222"
tibemsadmin -server "tcp://host:7222" -user admin -password secret
```

Some options are needed when you choose to make a SSL connection. For more information on SSL connections, refer to [Chapter 18, Using the SSL Protocol](#), page 465.

When You First Start tibemsadmin

The administration tool has a default user with the name `admin`. This is the default user for logging in to the administration tool.

To protect access to the server configuration, you must assign a password to the user `admin`.

Assign a Password to the Administrator

1. Log in and connect to the administration tool, as described directly above.
2. Use the `set password` command to change the password:

```
set password admin password
```

When you restart the administration tool and type `connect`, the administration tool now requires your password before connecting to the server.

For further information about setting and resetting passwords, refer to [set password on page 142](#).

Naming Conventions

These rules apply when naming users, groups, topics or queues:

- `$` is illegal at the beginning of the queue or topic names—but legal at the beginning of user and group names.
- A user name cannot contain colon (":") character.
- Space characters are permitted in a description field—if the entire description field is enclosed in double quotes (for example, "description field").
- Both `*` and `>` are wildcards, and cannot be used in names except as wildcards. For more information about wildcards, see [Wildcards on page 77](#).
- Dot separates elements within a destination name (**foo.bar.***) and can be used only for that purpose.

Name Length Limitations

The following length limitations apply for these parameter names:

- Destination name — cannot exceed 249 characters. For more information on topic and queue naming conventions, see [Destination Name Syntax on page 56](#).
- Username — cannot exceed 127 characters. The username parameter is described in [users.conf on page 262](#).
- Group name — cannot exceed 127 characters. The group-name parameter is described in [groups.conf on page 249](#).
- Client ID — cannot exceed 255 characters. The `clientID` parameter is described in [factories.conf on page 245](#).
- Connection URL — cannot exceed 1000 characters. The `url` parameter is described in [factories.conf on page 245](#).
- Passwords — cannot exceed 4096 characters. This length limitation applies to passwords used by the `tibemsd` to authenticate connecting clients or servers.

Command Listing

The command line interface of the administration tool allows you to perform a variety of functions. Note that when a system uses shared configuration files, the actions performed using the administration tool take effect only when connected to the active server.



Many of the commands listed below accept arguments that specify the names of users, groups, topics or queues. For information about the syntax and that apply to these names, see [Naming Conventions on page 127](#).



Note that SSL commands are not listed in this table. SSL commands are listed in several tables in [Chapter 18, Using the SSL Protocol, on page 465](#).

The following is an alphabetical listing of the commands including command syntax and a description of each command.

add member

```
add member group_name user_name [ ,user2,user3,...]
```

Add one or more users to the group. User names that are not already defined are added to the group as external users; see [Administration Commands and External Users and Groups on page 280](#).

addprop factory

```
addprop factory factory-name properties ...
```

Adds properties to the factory. Property names are separated by spaces.

See [factories.conf on page 245](#) for the list of factory properties.

An example is:

```
addprop factory MyTopicFactory ssl_trusted=cert1.pem
ssl_trusted=cert2.pem ssl_verify_host=disabled
```

addprop queue

```
addprop queue queue-name properties,...
```

Adds properties to the queue. Property names are separated by commas.

For information on properties that can be assigned to queues, see [Destination Properties on page 58](#).

addprop route

```
addprop route route-name prop=value[ prop-value...]
```

Adds properties to the route.

Destination (topic and queue) properties must be separated by commas but properties of routes and factories are separated with spaces.

You can set the `zone_name` and `zone_type` parameters when creating a route, but you cannot subsequently change them.

For route properties, see [Configuring Routes and Zones on page 518](#).

For the configuration file `routes.conf`, see [routes.conf on page 251](#).

addprop topic

```
addprop topic topic_name properties,...
```

Adds properties to the topic. Property names are separated by commas.

For information on properties that can be assigned to topics, see [Destination Properties on page 58](#).

autocommit

```
autocommit [on|off]
```

When `autocommit` is set to **on**, the changes made to the configuration files are automatically saved to disk after each command. When `autocommit` is set to **off**, you must manually use the `commit` command to save configuration changes to the disk.

By default, `autocommit` is set to **on** when interactively issuing commands.

Entering **autocommit** without parameters displays the current setting of `autocommit` (on or off).



Regardless of the `autocommit` setting, the EMS server acts on each admin command immediately making it part of the configuration. The `autocommit` feature only determines when the configuration is written to the files.

commit

```
commit
```

Commits all configuration changes into files on disk.

compact

```
compact store_name max_time
```

Compacts the store files for the specified store. Compaction is not available for stores of type `mstore`.

Since compaction can be a lengthy operation, and it blocks other database operations, *max_time* specifies a time limit (in seconds) for the operation. Note that *max_time* must be a number greater than zero.

If truncation is not enabled for the store file, the compact command will not reduce the file size. Truncation is enabled using the `file_truncate` parameter in the `stores.conf` file. See [stores.conf on page 253](#) for more information.

We recommend compacting the store files only when the Used Space usage is 30% or less (see [show store on page 169](#)).

connect

```
connect [server-url {admin|user_name} password]
```

Connects the administration tool to the server. Any administrator can connect. An administrator is either the `admin` user, any user in the `$admin` group, or any user that has administrator permissions enabled. See [Administrator Permissions on page 267](#) for more information about administrator permissions.

server-url is usually in the form:

```
protocol : //host-name : port-number
```

for example:

```
tcp : //myhost : 7222
```

The protocol can be `tcp` or `ssl`.

If a user name or password are not provided, the user is prompted to enter a user name and password, or only the password, if the user name was already specified in the command.

You can enter `connect` with no other options and the administrative tool tries to connect to the local server on the default port, which is 7222.

create bridge

```
create bridge source=type:dest_name target=type:dest_name
[selector=selector]
```

Creates a bridge between destinations.

type is either `topic` or `queue`.

For further information, see [bridges.conf on page 240](#).

create durable

```
create durable topic-name durable-name [property, ... ,property]
```

Creates a static durable subscriber.

For descriptions of parameters and properties, and information about conflict situations, see [durables.conf on page 244](#).

create factory

```
create factory factory_name factory_parameters
```

Creates a new connection factory.

For descriptions of factory parameters, see [factories.conf on page 245](#).

create group

```
create group group_name "description"
```

Creates a new group of users.

Initially, the group is empty. You can use the [add member](#) command to add users to the group.

create jndiname

```
create jndiname new_jndiname topic|queue|jndiname name
```

Creates a JNDI name for a topic or queue, or creates an alternate JNDI name for a topic that already has a JNDI name.

For example:

```
create jndiname FOO jndiname BAR
```

will create new JNDI name FOO referring the same object referred by JNDI name BAR

create queue

```
create queue queue_name [properties]
```

Creates a queue with the specified name and properties. The possible queue properties are described in [Destination Properties on page 58](#). Properties are listed in a comma-separated list, as described in [queues.conf on page 250](#).

create route

```
create route name url=URL [properties ...]
```

Creates a route.

The *name* must be the name of the other server to which the route connects.

The local server connects to the destination server at the specified URL. If you have configured fault-tolerant servers, you may specify the URL as a comma-separated list of URLs.

The route properties are listed in [routes.conf on page 251](#) and are specified as a space-separated list of parameter name and value pairs.

You can set the *zone_name* and *zone_type* parameters when creating a route, but you cannot subsequently change them.

If a passive route with the specified *name* already exists, this command promotes it to an *active-active* route; see [Active and Passive Routes on page 517](#).

For additional information on route parameters, see [Configuring Routes and Zones on page 518](#).

create rvcmlistener

```
create rvcmlistener transport_name cm_name subject
```

Registers an RVCML listener with the server so that any messages exported to a `tibrvcml` transport (including the first message sent) are guaranteed for the specified listener. This causes the server to perform the TIBCO Rendezvous call `tibrvcmlTransport_AddListener`.

The parameters are:

- *transport_name* — the name of the transport to which this RVCML listener applies.
- *cm_name* — the name of the RVCML listener to which topic messages are to be exported.
- *subject* — the RVCML subject name that messages are published to. This should be the same name as the topic names that specify the export property.

For more information, see [tibrvcm.conf on page 257](#) and [Rendezvous Certified Messaging \(RVCN\) Parameters on page 406](#).

create topic

```
create topic topic_name [properties]
```

Creates a topic with specified name and properties. See [Destination Properties on page 58](#) for the list of properties. Properties are listed in a comma-separated list, as described in [topics.conf on page 257](#).

create user

```
create user user_name ["user_description"] [password=password]
```

Creates a new user. Following the user name, you can add an optional description of the user in quotes. The password is optional and can be added later using the `set password` command.



User names cannot contain colon (:) characters.

delete all

```
delete all users|groups|topics|queues|durables  
[topic-name-pattern|queue-name-pattern]
```

If used as `delete all users|groups|topics|queues|durables` without the optional parameters, the command deletes all users, groups, topics, or queues (as chosen).

If used with a topic or queue, and the optional parameters, such as:

```
delete all topics|queues topic-name-pattern|queue-name-pattern
```

the command deletes all topics and queues that match the topic or queue name pattern.

delete bridge

```
delete bridge source=type:dest_name target=type:dest_name
```

Delete the bridge between the specified source and target destinations.

type is either `topic` or `queue`.

See [Destination Bridges on page 82](#) for more information on bridges.

delete connection

```
delete connection connection-id
```

Delete the named connection for the client. The connection ID is shown in the first column of the connection description printed by `show connection`.

delete durable

```
delete durable durable-name clientID
```

Delete the named durable subscriber.

When both the durable name and the client ID are specified, the EMS Server looks for a durable named *clientID:durable-name* in the list of durables. If a matching durable subscriber is not found, the administration tool prints an error message including the fully qualified durable name.

See also, [Conflicting Specifications on page 244](#).

delete factory

```
delete factory factory-name
```

Delete the named connection factory.

delete group

```
delete group group-name
```

Delete the named group.

delete jndiname

```
delete jndiname jndiname
```

Delete the named JNDI name. Notice that deleting the last JNDI name of a connection factory object will remove the connection factory object as well.

See [Chapter 12, Using the EMS Implementation of JNDI, on page 359](#) for more information.

delete message

```
delete message messageID
```

Delete the message with the specified message ID.

delete queue

```
delete queue queue-name
```

Delete the named queue.

delete route

```
delete route route-name
```

Delete the named route.

delete rvcmlistener

```
delete rvcmlistener transport_name cm_name subject
```

Unregister an RVCML listener with the server so that any messages being held for the specified listener in the RVCML ledger are released. This causes the server to perform the TIBCO Rendezvous call `tibrvcmlTransport_RemoveListener`.

The parameters are:

- *transport_name* — the name of the transport to which this RVCML listener applies.
- *cm_name* — the name of the RVCML listener to which topic messages are exported.
- *subject* — the RVCML subject name that messages are published to. This should be the same name as the topic names that specify the `export` property.

For more information, see [tibrvcml.conf on page 257](#) and [Rendezvous Certified Messaging \(RVCML\) Parameters on page 406](#).

delete topic

```
delete topic topic-name
```

Delete the named topic.

delete user

```
delete user user-name
```

Delete the named user.

disconnect

```
disconnect
```

Disconnect the administrative tool from the server.

echo

```
echo [on|off]
```

Echo controls the reports that are printed into the standard output. When `echo` is `off` the administrative tool only prints errors and the output of queries. When `echo` is `on`, the administrative tool report also contains a record of successful command execution.

Choosing the parameter `on` or `off` in this command controls `echo`. If `echo` is entered in the command line without a parameter, it displays the current `echo` setting (`on` or `off`). This command is used primarily for scripts.

The default setting for `echo` is `on`.

exit

```
exit (aliases: quit, q, bye, end)
```

Exit the administration tool.

The administrator may choose the `exit` command when there are changes in the configuration have which have not been committed to disk. In this case, the system will prompt the administrator to use the `commit` command before exiting.

grant queue

```
grant queue queue-name user=name | group=name permissions
```

Grants specified permissions to specified user or group on specified queue. The name following the queue name is first checked to be a group name, then a user name.

Specified permissions are added to any existing permissions. Multiple permissions are separated by commas. Enter `all` in the *permissions* string if you choose to grant all possible user permissions.

User permissions are:

- `receive`
- `send`
- `browse`

For more information on queue permissions, see [Table 48](#) in [User Permissions on page 283](#).

Destination-level administrator permissions can also be granted with this command. The following are administrator permissions for queues.

- view
- create
- delete
- modify
- purge

For more information on destination permissions, see [Destination-Level Permissions on page 272](#).

grant topic

```
grant topic topic-name user=name | group=name permissions
```

Grants specified permissions to specified user or group on specified topic. The name following the topic name is first checked to be a group name, then a user name.

Specified permissions are added to any existing permissions. Multiple permissions are separated by commas. Enter **all** in the *permissions* string if you choose to grant all possible permissions.

Topic permissions are:

- subscribe
- publish
- durable
- use_durable

For more information on topic permissions, see [Table 49](#) in [User Permissions on page 283](#).

Destination-level administrator permissions can also be granted with this command. The following are administrator permissions for topics.

- view
- create
- delete

- `modify`
- `purge`

For more information on destination permissions, see [Destination-Level Permissions on page 272](#).

grant admin

```
grant admin user=name | group=name admin_permissions
```

Grant the named global administrator permissions to the named user or group. For a complete listing of global administrator permissions, see [Global Administrator Permissions on page 269](#).

help

```
help (aliases: h, ?)
```

Display help information.

Enter `help commands` for a summary of all available commands.

Enter `help command` for help on a specific command.

info

```
info (alias: i)
```

Shows server name and information about the connected server.

jaci clear

```
jaci clear
```

Empties the JACI permission cache of all entries.

jaci resetstats

```
jaci resetstats
```

Resets all statistics counters for the JACI cache to zero.

jaci showstats

```
jaci showstats
```

Prints statistics about JACI cache performance.

purge all queues

```
purge all queues [pattern]
```

Purge all or selected queues.

When used without the optional pattern parameter, this command erases all messages in all queues for all receivers.

When used with the *pattern* parameter, this command erases all messages in all queues that fit the pattern (for example: `foo.*`).

purge all topics

```
purge all topics [pattern]
```

Purge all or selected topics.

When used without the optional pattern parameter, this command erases all messages in all topics for all subscribers.

When used with the *pattern* parameter, this command erases all messages in all topics that fit the pattern (for example: `foo.*`).

purge durable

```
purge durable durable-name
```

Purge all messages in the topic for the named durable subscriber

purge queue

```
purge queue queue-name
```

Purge all messages in the named queue.

purge topic

```
purge topic topic-name
```

Purge all messages for all subscribers on the named topic.

remove member

```
remove member group-name user-name[, user2, user3, . . .]
```

Remove one or more named users from the named group.

removeprop factory

```
removeprop factory factory-name properties
```

Remove the named properties from the named factory. See [Connection Factory Parameters on page 245](#) for a list of properties.

removeprop queue

```
removeprop queue queue-name properties
```

Remove the named properties from the named queue.

removeprop route

```
removeprop route route-name properties
```

Remove the named properties from the named route.

You cannot remove the URL.

You can set the `zone_name` and `zone_type` parameters when creating a route, but you cannot subsequently change them.

For route parameters, see [Configuring Routes and Zones on page 518](#).

For the configuration file `routes.conf`, see [routes.conf on page 251](#).

removeprop topic

```
removeprop topic topic-name properties
```

Remove the named properties from the named topic.

resume route

```
resume route route-name
```

Resumes sending messages to named route, if messages were previously suspended using the [suspend route](#) command.

revoke admin

```
revoke admin user=name | group=name permissions
```

Revoke the specified global administrator permissions from the named user or group. See [Chapter 8, Authentication and Permissions, on page 265](#) for more information about administrator permissions.

revoke queue

```
revoke queue queue-name user=name | group=name permissions
revoke queue queue-name * [user | admin | both]
```

Revoke the specified permissions from a user or group for the named queue.

User and group permissions for queues are `receive`, `send`, `browse`, and `all`. Administrator permissions for queues are `view`, `create`, `delete`, `modify`, and `purge`.

If you specify an asterisk (*), all user-level permissions on this queue are removed. You can use the optional `admin` parameter to revoke all administrative permissions, or the `both` parameter to revoke all user-level and administrative permissions on the queue.

For more information, see [Chapter 8, Authentication and Permissions, on page 265](#).

revoke topic

```
revoke topic topic-name user=name | group=name permissions
revoke topic topic-name * [user | admin | both]
```

Revoke the specified permissions from a user or group for the named topic.

User and group permissions for topics are `subscribe`, `publish`, `durable`, `use_durable`, and `all`. Administrator permissions for topics are `view`, `create`, `delete`, `modify`, and `purge`.

If you specify an asterisk (*), all user-level permissions on this topic are removed. You can use the optional `admin` parameter to revoke all administrative permissions, or the `both` parameter to revoke all user-level and administrative permissions on the topic.

For more information, see [Chapter 8, Authentication and Permissions, on page 265](#).

rotatelog

```
rotatelog
```

Force the current log file to be backed up and truncated. The server starts writing entries to the newly empty log file.

The backup file name is the same as the current log file name with a sequence number appended to the filename. The server queries the current log file directory and determines what the highest sequence number is, then chooses the next highest sequence number for the new backup name. For example, if the log file name is `tibems.log` and there is already a `tibems.log.1` and `tibems.log.2`, the server names the next backup `tibems.log.3`.

set password

```
set password user-name [password]
```

Set the password for the named user.

If you do not supply a password in the command, the server prompts you to type one.

- To reset a password, type:

```
set password user-name
```

Type a new password at the prompt.

- To remove a password, use this command without supplying a password, and press the **Enter** key at the prompt (without typing a password).



Passwords are a significant point of vulnerability for any enterprise. We recommend enforcing strong standards for passwords.

For security equivalent to single DES (an industry minimum), security experts recommend passwords that contain 8–14 characters, with at least one upper case character, at least one numeric character, and at least one punctuation character.

set server

```
set server parameter=value [parameter=value ...]
```

The `set server` command can control many parameters. Multiple parameters are separated by spaces. [Table 16](#) describes the parameters you can set with this command.

Table 16 `set server` — parameters (Sheet 1 of 6)

Parameter	Description
<code>password [= string]</code>	<p>Sets server password used by the server to connect to other routed servers. If the value is omitted it is prompted for by the administration tool. Entered value will be stored in the main server configuration file in mangled form (but not encrypted).</p> <p>To reset this password, enter the empty string twice at the prompt.</p>
<code>authorization=enabled disabled</code>	<p>Sets the authorization mode in the tibemsd.conf file.</p> <p>After a transition from disabled to enabled, the server checks ACL permissions for all subsequent requests. While the server requires valid authentication for existing producers and consumers, it does not retroactively reauthenticate them; it denies access to users without valid prior authentication.</p>

Table 16 *set server — parameters (Sheet 2 of 6)*

Parameter	Description
<code>log_trace=trace-items</code>	<p>Sets the trace preference on the file defined by the <code>logfile</code> parameter. If <code>logfile</code> is not set, the values are stored but have no effect.</p> <p>The value of this parameter is a comma-separated list of trace options. For a list of trace options and their meanings, see Table 72, Server Tracing Options, on page 448.</p> <p>You may specify trace options in three forms:</p> <ul style="list-style-type: none">• <code>plain</code> A trace option without a prefix character replaces any existing trace options.• <code>+</code> A trace option preceded by <code>+</code> adds the option to the current set of trace options.• <code>-</code> A trace option preceded by <code>-</code> removes the option from the current set of trace options. <p>Examples</p> <p>The following example sets the trace log to only show messages about access control violations.</p> <pre>log_trace=ACL</pre> <p>The next example sets the trace log to show all default trace messages, in addition to SSL messages, but ADMIN messages are not shown.</p> <pre>log_trace=DEFAULT, -ADMIN, +SSL</pre>

Table 16 *set server — parameters (Sheet 3 of 6)*

Parameter	Description
<code>console_trace=console-trace-items</code>	<p>Sets trace options for output to <code>stderr</code>. The values are the same as for <code>log_trace</code>. However, console tracing is independent of log file tracing.</p> <p>If <code>logfile</code> is defined, you can stop console output by specifying:</p> <pre>console_trace=-DEFAULT</pre> <p>Note that important error messages (and some other messages) are always output, overriding the trace settings.</p> <p>Examples</p> <p>This example sends a trace message to the console when a TIBCO Rendezvous advisory message arrives.</p> <pre>console_trace=RVADV</pre>
<pre>client_trace={enabled disabled} [target=location] [filter=value]</pre>	<p>Administrators can trace a connection or group of connections. When this property is enabled, the client generates trace output for opening or closing a connection, message activity, and transaction activity. This type of tracing does not require restarting the client program.</p> <p>The client sends trace output to <i>location</i>, which may be either <code>stderr</code> (the default) or <code>stdout</code>.</p> <p>You can specify a filter to selectively trace specific connections. The <i>filter</i> can be <code>user</code>, <code>connid</code> or <code>clientid</code>. The <i>value</i> can be a user name or ID (as appropriate to the filter).</p> <p>When the filter and value clause is absent, the default behavior is to trace all connections.</p> <p>Setting this parameter using the administration tool does not change its value in the configuration file <code>tibemsd.conf</code>.</p>

Table 16 *set server — parameters (Sheet 4 of 6)*

Parameter	Description
<code>max_msg_memory=value</code>	<p>Maximum memory the server can use for messages.</p> <p>For a complete description, see <code>max_msg_memory</code> in tibemsd.conf on page 187.</p> <p>Specify units as KB, MB or GB. The minimum value is 8MB. Zero is a special value, indicating no limit.</p> <p>Lowering this value will not immediately free memory occupied by messages.</p>
<code>msg_swapping=enabled disabled</code>	Enables or disables the ability to swap messages to disk.
<code>track_message_ids=enabled disabled</code>	Enables or disables tracking messages by MessageID.
<code>track_correlation_ids=enabled disabled</code>	Enables or disables tracking messages by CorrelationID.
<code>ssl_password[=string]</code>	<p>This sets a password for SSL use only.</p> <p>Sets private key or PKCS#12 file password used by the server to decrypt the content of the server identity file. The password is stored in mangled form.</p>
<code>ft_ssl_password[=string]</code>	<p>This sets a password for SSL use with Fault Tolerance.</p> <p>Sets private key or PKCS#12 file password used by the server to decrypt the content of the FT identity file. The password is stored in mangled form.</p>

Table 16 *set server — parameters (Sheet 5 of 6)*

Parameter	Description
<code>server_rate_interval=num</code>	<p>Sets the interval (in seconds) over which overall server statistics are averaged. This parameter can be set to any positive integer greater than zero.</p> <p>Overall server statistics are always gathered, so this parameter cannot be set to zero. By default, this parameter is set to 1.</p> <p>Setting this parameter allows you to average message rates and message size over the specified interval.</p>
<code>statistics=enabled disabled</code>	<p>Enables or disables statistic gathering for producers, consumers, destinations, and routes. By default this parameter is set to disabled.</p> <p>Disabling statistic gathering resets the total statistics for each object to zero.</p>
<code>rate_interval=num</code>	<p>Sets the interval (in seconds) over which statistics for routes, destinations, producers, and consumers are averaged. By default, this parameter is set to 3 seconds. Setting this parameter to zero disables the average calculation.</p>
<code>detailed_statistics=NONE PRODUCERS, CONSUMERS, ROUTES, CHANNELS</code>	<p>Specifies which objects should have detailed statistic tracking. Detailed statistic tracking is only appropriate for routes, channels, producers that specify no destination, or consumers that specify wildcard destinations. When detailed tracking is enabled, statistics for each destination are kept for the object.</p> <p>Setting this parameter to NONE disables detailed statistic tracking. You can specify any combination of PRODUCERS, CONSUMERS, ROUTES, or CHANNELS to enable tracking for each object. If you specify more than one type of detailed tracking, separate each item with a comma.</p>

Table 16 *set server — parameters (Sheet 6 of 6)*

Parameter	Description
<code>statistics_cleanup_interval=num</code>	Specifies how long (in seconds) the server should keep detailed statistics if the destination has no activity. This is useful for controlling the amount of memory used by detailed statistic tracking. When the specified interval is reached, statistics for destinations with no activity are deleted.
<code>max_stat_memory=num</code>	<p>Specifies the maximum amount of memory to use for detailed statistic gathering. If no units are specified, the amount is in bytes, otherwise you can specify the amount using KB, MB, or GB as the units.</p> <p>Once the maximum memory limit is reached, the server stops collecting detailed statistics. If statistics are deleted and memory becomes available, the server resumes detailed statistic gathering.</p>

setprop factory

`setprop factory factory-name properties ...`

Set the properties for a connection factory, overriding any existing properties. Multiple properties are separated by spaces. See [Connection Factory Parameters on page 245](#) for the list of the properties that can be set for a connection factory.

setprop queue

`setprop queue queue-name properties, ...`

Set the properties for a queue, overriding any existing properties. Any properties on a queue that are not explicitly specified by this command are removed.

Multiple properties are separated by commas. See [Destination Properties on page 58](#) for the list of the properties that can be set for a queue.

setprop route

```
setprop route route-name properties ...
```

Set the properties for a route, overriding any existing properties. Any properties on a route that are not explicitly specified by this command are removed.

You can set the `zone_name` and `zone_type` parameters when creating a route, but you cannot subsequently change them.

Multiple properties are separated by spaces. For route parameters, see [routes.conf on page 251](#) and [Configuring Routes and Zones on page 518](#).

setprop topic

```
setprop topic topic-name properties
```

Set topic properties, overriding any existing properties. Any properties on a topic that are not explicitly specified by this command are removed.

Multiple properties are separated by commas. See [Destination Properties on page 58](#) for the list of the properties that can be set for a topic.

show bridge

```
show bridge topic|queue bridge_source
```

Display information about the configured bridges for the named topic or queue. The `bridge_source` is the name of the topic or queue established as the source of the bridge.

The following is example output for this command:

Target Name	Type	Selector
queue.dest	Q	
topic.dest.1	T	"urgency in ('high', 'medium')"
topic.dest.2	T	

The names of the destinations to which the specified destination has configured bridges are listed in the Target Name column. The type and the message selector (if one is defined) for the bridge are listed in the Type and Selector column.

show bridges

```
show bridges [type=topic|queue] [pattern]
```

Shows a summary of the destination bridges that are currently configured. The *type* option specifies the type of destination established as the bridge source. For example, `show bridges topic` shows a summary of configured bridges for all topics that are established as a bridge source. The *pattern* specifies a pattern to match for source destination names. For example `show bridges foo.*` returns a summary of configured bridges for all source destinations that match the name `foo.*`. The *type* and *pattern* are optional.

The following is example output for this command:

Source Name	Queue Targets	Topic Targets
Q queue.source	1	1
T topic.source	1	2

Destinations that match the specified pattern and/or type are listed in the Source Name column. The number of bridges to queues for each destination is listed in the Queue Targets column. The number of bridges to topics for each destination is listed in the Topic Targets column.

show channel

```
show channel channel-name
```

Show the details of a specific multicast channel. The *channel-name* must be the exact name of a specific channel. Wildcards and partial names are invalid.

This command prints a table of information described in [Table 17](#).

Table 17 *show channel* — description of output fields

Heading	Description
Channel	Name of the multicast channel.
Address	The multicast group IP address and port destination to which messages are broadcast, in the form: <i><multicast-group-IP-address> : <multicast-port></i>
TTL	The maximum number of number of network hops allowed for data on the channel.
Priority	The transmission priority of messages on this channel when the EMS server allocates bandwidth. The highest priority is -5 and the lowest is 5.

Table 17 *show channel* — description of output fields

Heading	Description
Max Rate	The maximum rate at which the server broadcasts messages over the channel.
Max Time	The maximum length of time, in seconds, that the server holds sent messages for retransmission.
Interface	The IP address over which the server sends multicast traffic on this channel. A value of 0.0.0.0 indicates that the default interfaces is being used.
Status	The status of the channel, either <i>active</i> or <i>inactive</i> .
Server Backlog	The number of messages and the total number of bytes pending broadcast over the channel. See Multicast and Flow Control on page 88 for more information about controlling backlog.
Transmitted	The total number of bytes sent on the channel. This number does not include retransmissions.
Retransmitted	The total number of bytes sent in retransmissions on the channel.
Retransmission Buffer	The total number of bytes that are currently buffered for retransmission.

show channels

```
show channels
```

Print a summary of the server's multicast channels, including each channel's multicast address and status.

show config

```
show config
```

Shows the configuration parameters for the connected server. The output includes:

- configuration files
- server database
- server JVM
- server JDBC database
- listen ports
- configuration settings
- message tracking
- server tracing parameters
- statistics settings
- fault-tolerant setup
- external transport setup
- server SSL setup

show consumer

```
show consumer consumerID
```

Shows details about a specific consumer. The *consumerID* can be obtained from the [show consumers](#) output.

show consumers

```
show consumers [topic=name | queue=name] [durable] [user=name]  
[connection=id] [sort=conn|user|dest|msgs] [full]
```

Shows information about all consumers or only consumers matching specified filters. Output of the command can be controlled by specifying the `sort` or `full` parameter. If the `topic` or `queue` parameter is specified, then only consumers on destinations matching specified queue or topic are shown. The `user` and/or `connection` parameters show consumers only for the specified user or connection. Note that while the queue browser is open, it appears as a consumer in the EMS server.

The `durable` parameter shows only durable topic subscribers and queue receivers, but it does not prevent queue consumers to be shown. To see only durable topic consumers, use:

```
show consumers topic=> durable
```

The `sort` parameter sorts the consumers by either connection ID, user name, destination name, or number of pending messages. The `full` parameter shows all columns listed below and can be as wide as 120-140 characters or wider. Both topic and queue consumers are shown in separate tables, first the topic consumers and then the queue consumers.



When connected to an EMS 8.0 server, this command no longer displays offline durable subscribers. In order to see offline durables, use the command `show durables` or `show subscriptions` .

Table 18 *show consumers* — description of output fields

Heading	Description
Id	Consumer ID.
Conn	Consumer's connection ID. If performed on an EMS 7.x or earlier server, this field displays '-' to indicate a disconnected durable topic subscriber.
Sess	Consumer's session ID. If performed on an EMS 7.x or earlier server, this field displays '-' to indicate a disconnected durable topic subscriber.

Table 18 *show consumers* — description of output fields

Heading	Description
T	<p>Consumer type character which can be one of:</p> <p>For topic consumer:</p> <ul style="list-style-type: none">• T - non-durable topic subscriber.• D - durable topic subscriber.• R - system-created durable for a routed topic.• P - proxy subscriber on route's temporary topic. <p>For queue consumer:</p> <ul style="list-style-type: none">• Q - regular queue receiver.• q - inactive queue receiver.• P - system-created receiver on global queue for user receiver created in one of routes.
Topic/Queue	Name of the subscription topic or queue.
Name	(Topics Only.) Durable or shared subscription name. This column is shown for topic consumers if at least one consumer is a durable or shared consumer.

Table 18 *show consumers* — description of output fields

Heading	Description
SAS[NMBS]	<p>Description of columns:</p> <ul style="list-style-type: none"> • S - '+' if consumer's connection started, '-' otherwise. • A - mode of consumer's session, values are: <ul style="list-style-type: none"> — N - no acknowledge — A - auto acknowledge — D - dups_ok acknowledge — C - client acknowledge — T - session is transactional — X - XA or MS DTC session — Z - connection consumer • S - '+' if consumer has a selector, '-' otherwise. • N - (TOPICS ONLY) '+' if subscriber is "NoLocal." • M - (TOPICS ONLY) '+' if subscriber is receiving multicast. • B - (QUEUES ONLY) '+' if consumer is a queue browser. • S - (TOPICS ONLY) '+' if this is a shared consumer.
Pre	Prefetch value of the consumer's destination.
Pre Dlv	Number of prefetch window messages delivered to consumer
Msgs Sent	Current number of messages sent to consumer which are not yet acknowledged by consumer's session.
Size Sent	Combined size of unacknowledged messages currently sent to consumer. Value is rounded and shown in bytes, (K)ilobytes, (M)egabytes or (G)igabytes.
Pend Msgs	(Topics Only.) Total number of messages pending for the topic consumer.
Pend Size	(Topics Only.) Combined size of messages pending for the topic consumer. Value is rounded and shown in bytes, (K)ilobytes, (M)egabytes or (G)igabytes.
Uptime	Uptime of the consumer.
Last Sent	Approximate time elapsed since last message was sent by the server to the consumer. Value is approximate with precision of 1 second.

Table 18 *show consumers* — description of output fields

Heading	Description
Last Akcd	Approximate time elapsed since last time a message sent to the consumer was acknowledged by consumer's session. Value is approximate with precision of 1 second.
Total Sent	Total number of messages sent to consumer since it was created. This includes resends due to session recover or rollback.
Total Acked	Total number of messages sent to the consumer and acknowledged by consumer's session since consumer created.

show connections

```
show connections [type=q|t|s] [host=hostname] [user=username]
[version] [address] [counts] [full]
```

Show connections between clients and server. [Table 20 on page 157](#) describes the output.

The `type` parameter selects the subset of connections to display as shown in [Table 19](#). The `host` and `user` parameters can further narrow the output to only those connections involving a specific host or user. When the `version` flag is present, the display includes the client's version number.

If the `address` parameter is specified, then the IP address is printed in the output table. If the `counts` parameter is specified, then number of producers, consumers and temporary destinations are printed. Specifying the `full` parameter prints all of the available information.

Table 19 *show connections* — type parameter

Type	Description
type=q	Show queue connections only.
type=t	Show topic connections only.
type=s	Show system connections only.
absent	Show queue and topic connections, but not system connections.

Table 20 *show connections* — description of output fields

Heading	Description
L	<p>The type of client. Can be one of the following:</p> <ul style="list-style-type: none"> • J — Java client • c — C client • # — C# client • - — unknown system connection
Version	The EMS version of the client.
ID	Unique connection ID. Each connection is assigned a unique, numeric ID that can be used to delete the connection.
FSXT	<p>Connection type information.</p> <p>The F column displays whether the connection is fault-tolerant.</p> <ul style="list-style-type: none"> • - — not a fault-tolerant connection, that is, this connection has no alternative URLs • + — fault-tolerant connection, that is, this connection has alternative URLs <p>The S column displays whether the connection uses SSL.</p> <ul style="list-style-type: none"> • - — connection is not SSL • + — connection is SSL <p>The X column displays whether the connection is an XA or MS DTC transaction.</p> <ul style="list-style-type: none"> • - — connection is not XA or MS DTC • + — connection is either an XA or MS DTC connection <p>The T column displays the connection type.</p> <ul style="list-style-type: none"> • C — generic user connection • T — user TopicConnection • Q — user QueueConnection • A — administrative connection • R — system connection to another route server • F — system connection to the fault-tolerant server

Table 20 *show connections* — description of output fields

Heading	Description
S	Connection started status, + if started, - if stopped.
IP Address	Shows client IP address. The address or full parameter must be specified to display this field.
Host	Connection's host name. (If the name is not available, this column displays the host's IP address.)
Address	Connection's IP address. If you supply the keyword address, then the table includes this column.
User	Connection user name. If a user name was not provided when the connection was created, it is assigned the default user name anonymous.
ClientID	Client ID of the connection.
Sess	Number of sessions on this connection.
Prod	Number of producers on this connection. The counts or full parameter must be specified to display this field.
Cons	Number of consumers on this connection. The counts or full parameter must be specified to display this field.
TmpT	Number of temporary topics created by this connection. For clients prior to 4.4 this is not known and shows "?." The counts or full parameter must be specified to display this field.
TmpQ	Number of temporary queues created by this connection. For clients prior to 4.4 this is not known and shows "?." The counts or full parameter must be specified to display this field.

Table 20 *show connections* — description of output fields

Heading	Description
Uncomm	Number of messages in uncommitted transactions on the connection. The <code>counts</code> or <code>full</code> parameter must be specified to display this field.
UncommSize	The combined size, in bytes, of messages in uncommitted transactions on the connection. The <code>counts</code> or <code>full</code> parameter must be specified to display this field.
Uptime	Time that the connection has been in effect.

show db

`show db`

Print a summary of the server's databases. Databases are also printed by [show stores](#), the preferred command.

See the [show store on page 169](#) for details about a specific database.

show durable

`show durable durable-name`

Show information about a durable subscriber.

Table 21 *show durable* — table Information

Heading	Description
Durable Subscriber	Fully qualified name of the durable subscriber. This name concatenates the client ID (if any) and the subscription name (separated by a colon).
Subscription name	Full name of the durable subscriber.
Shared	yes if this is a shared durable subscription, no otherwise.
Client ID	Client ID of the subscriber's connection.

Table 21 *show durable* — table Information (Cont'd)

Heading	Description
Topic	The topic from which the durable subscription receives messages.
Type	dynamic—created by a client static—configured by an administrator
Status	online offline
Username	Username of the durable subscriber (that is, of the client's connection). If the durable subscriber is currently offline, the value in this column is offline.
Consumer ID	This internal ID number is not otherwise available outside the server.
No Local	enabled—the subscriber does not receive messages sent from its local connection (that is, the same connection as the subscriber). disabled—the subscriber receives messages from all connections.
Selector	The subscriber receives only those messages that match this selector.
Pending Msgs	Number of all messages in the topic. (This count includes the number of delivered messages.)
Delivered Msgs	Number of messages in the topic that have been delivered to the durable subscriber, but not yet acknowledged.
Pending Msgs Size	Total size of all pending messages

show durables

`show durables [pattern]`

If a pattern is not entered, this command shows a list of all durable subscribers on all topics.

If a pattern is entered (for example `foo.*`) this command shows a list of durable subscribers on topics that match that pattern.

This command prints a table of information described in [Table 22](#).

Table 22 *show durables* — table Information

Heading	Description
Topic Name	Name of the topic. An asterisk preceding this name indicates a dynamic durable subscriber. Otherwise the subscriber is static (configured by an administrator).
Durable	Full name of the durable subscriber.
Shared	Y to indicate that this is a shared durable subscription, N otherwise.
User	Name of the user of this durable subscriber. If the durable subscriber is currently offline, the value in this column is <code>offline</code> . If this is a shared durable subscription, the value of this column is <code>shared</code> . For users defined externally, there is an asterisk in front of the user name.
Msgs	Number of pending messages
Size	Total size of pending messages

For more information, see [Destination Properties on page 58](#).

show factory

```
show factory factory-name
```

Shows properties of specified factory.

show factories

```
show factories [generic|topic|queue]
```

Shows all factories. You can refine the listed output by specifying only generic, topic, or queue factories be listed.

show jndiname

```
show jndiname jndi-name
```

Shows the object that the specified name is bound to by the JNDI server.

show jndinames

```
show jndinames [type]
```

The optional parameter *type* can be:

- destination
- topic
- queue
- factory
- topicConnectionFactory
- queueConnectionFactory

When *type* is specified only JNDI names bound to objects of the specified type are shown. When *type* is not specified, all JNDI names are shown.

show group

```
show group group-name
```

Shows group name, description, and number of members in the group.

For groups defined externally, there is an asterisk in front of the group name. Only external groups with at least one currently connected user are shown.

show groups

```
show groups
```

Shows all user groups.

For groups defined externally, there is an asterisk in front of the group name.

show members

```
show members group-name
```

Shows all user members of specified user group.

show message

```
show message messageID
```

Shows the message for the specified message id.

This command requires that tracking by message ID be turned on using the `track_message_ids` configuration parameter.

show messages

```
show messages correlationID
```

Shows the message IDs of all messages with the specified correlation ID set as `JMSCorrelationID` message header field. You can display the message for each ID returned by this command by using the [show message](#) *messageID* command.

This command requires that tracking by correlation ID be turned on using the `track_correlation_ids` configuration parameter.

show parents

```
show parents user-name
```

Shows the user's parent groups. This command can help you to understand the user's permissions.

show queue

```
show queue queue-name
```

Shows the details for the specified queue.



If the queue is a routed queue, specify only the name of the queue (do not specify the server using the *queue-name@server* form).

Table 23 *show queue* — table Information

Heading	Description
Queue	Full name of the queue.
Type	dynamic—created by a client static—configured by an administrator
Properties	A list of property names that are set on the queue, and their values. For an index list of property names, see Destination Properties on page 58 .
JNDI Names	A list of explicitly assigned JNDI names that refer to this queue.
Bridges	A list of bridges from this queue to other destinations.
Receivers	Number of consumers on this queue.
Pending Msgs	Number of all messages in the queue, followed by the number of persistent messages in parenthesis. These counts include the number of delivered messages.
Delivered Msgs	Number of messages in the queue that have been delivered to a consumer, but not yet acknowledged.
Pending Msgs Size	Total size of all pending messages, followed by the size of all persistent messages in parenthesis.

show queues

```
show queues [pattern-name [notemp|static|dynamic]
[first=n|next=n|last=n]]
```

If a *pattern-name* is not entered, this command shows a list of all queues.

If a *pattern-name* is entered (for example `foo.*` or `foo.>`) this command shows a list of queues that match that pattern. See [Wildcards * and > on page 77](#) for more information about using wildcards.

You can further refine the list of queues that match the pattern by using one of the following parameters:

- `notemp` — do not show temporary queues
- `static` — show only static queues
- `dynamic` — show only dynamic queues

When a *pattern-name* is entered, you can also cursor through the list of queues using one of the following commands, where *n* is whole number:

- `first=n` — show the first *n* queues
- `next=n` — show the next *n* queues
- `last=n` — show the next *n* queues and terminate the cursor

The cursor examines *n* queues and displays queues that match the *pattern-name*. Because it does not traverse the full list of queues, the cursor may return zero or fewer than *n* queues. To find all matching queues, continue to use `next` until you receive a `Cursor complete` message.

The `show queues` command prints a table of information described in [Table 24](#). A * appearing before the queue name indicates a dynamic queue.

Table 24 *show queues* — table Information

Heading	Description
Queue Name	Name of the queue. If the name is prefixed with an asterisk (*), then the queue is temporary or was created dynamically. Properties of dynamic and temporary queues cannot be changed.

Table 24 *show queues — table Information (Cont'd)*

Heading	Description
SNFGXIBCT	Prints information on the topic properties in the order (S)ecure (N)sender_name or sender_name_enforced (F)ailsafe (G)lobal e(X)clusive (I)mport (B)ridge (C)flowControl (T)race The characters in the value section show: - Property not present + Property is present, and was set on the topic itself * Property is present, and was inherited from another queue Note that inherited properties cannot be removed.
Pre	Prefetch value. If the value is followed by an asterisk (*), then it is inherited from another queue or is the default value.
Rcvrs	Number of currently active receivers
All Msgs	
Msgs	Number of pending messages
Size	Total size of pending messages
Persistent Msgs	
Msgs	Number of pending persistent messages
Size	Total size of pending persistent messages

For more information, see [Destination Properties on page 58](#).

show route

```
show route route-name
```

Shows the properties (URL and SSL properties) of a route.

show routes

```
show routes
```

Shows the properties (URL and SSL properties) of all created routes.

These commands print the information described in [Table 25](#).

Table 25 *show routes* — table Information

Heading	Description
Route	Name of the route.
T	Type of route: <ul style="list-style-type: none"> • A indicates an active route. • P indicates a passive route.
ConnID	Unique ID number of the connection from this server to the server at the other end of the route. A hyphen (-) in this column indicates that the other server is not connected.
URL	URL of the server at the other end of the route.
ZoneName	Name of the zone for the route.
ZoneType	Type of the zone: <ul style="list-style-type: none"> • m indicates a multi-hop zone. • 1 indicates a one-hop zone.

show rvctransportledger

```
show rvctransportledger transport_name [subject-or-wildcard]
```

Displays the TIBCO Rendezvous certified messaging (RVCN) ledger file entries for the specified transport and the specified subject. You can specify a subject name, use wildcards to retrieve all matching subjects, or omit the subject name to retrieve all ledger file entries.

For more information about ledger files and the format of ledger file entries, see TIBCO Rendezvous documentation.

show rvcmlisteners

```
show rvcmlisteners
```

Shows all RVCML listeners that have been created using the `create rvcmlistener` command or by editing the `tibbrvcml.conf` file.

show server

```
show server (aliases: info, i)
```

Shows server name and information about the connected server.

show stat

```
show stat channel name [topic=name]
```

```
show stat consumers [topic=name|queue=name] [user=name]  
[connection=id] [total]
```

```
show stat producers [topic=name|queue=name] [user=name]  
[connection=id] [total]
```

```
show stat route name [topic=name|queue=name] [total] [wide]
```

```
show stat topic name [total] [wide]
```

```
show stat queue name [total] [wide]
```

Displays statistics for the specified item. You can display statistics for consumers, producers, routes, destinations, or channels. Statistic gathering must be enabled for statistics to be displayed. Also, detailed statistics for each item can be displayed if detailed statistic tracking is enabled. Averages for inbound/outbound messages and message size are available if an interval is specified in the `rate_interval` configuration parameter.

The `total` keyword specifies that only total number of messages and total message size for the item should be displayed. The `wide` keyword displays inbound and outbound message statistics on the same line.

See [Working with Server Statistics on page 460](#) for a complete description of statistics and how to enable/disable statistic gathering options.



When connected to an EMS 8.0 server, this command does not return statistics for offline durable subscribers.

show store

```
show store store-name
```

Show the details of a specific store. This command can be used to get details about either a file-based store or a database store.

The *store-name* must be the exact name of a specific store.

This command prints a table of information described in [Table 26](#).

Table 26 show store — table Information

Heading	Description
Store	Name of the store.
Type	Type of store: <ul style="list-style-type: none"> • file indicates a file-based store. • dbstore indicates a database store. • mstore indicates an mstore.
Message Count	The number of messages that are stored in the file.
Swapped Count	The number of messages that have been swapped from process memory to store file.
Average Write Time	Average time in seconds a write call takes. (Not available for asynchronous file stores.)
Write Usage	The ratio between time spent within write calls and the time specified by the server_rate_interval . (Not available for asynchronous file stores.)
Headings specific to file-based stores	
File	File name associated with this store file, as it is set by the file parameter in the stores.conf file.
Access Mode	asynchronous—the server stores messages in the file using asynchronous I/O calls. synchronous—the server stores messages in the file using synchronous I/O calls.
Pre-allocation Minimum	The amount of disk space, if any, that is preallocated to this file.

Table 26 *show store — table Information*

Heading	Description
CRC	<p>enabled—the server uses CRC to validate checksum data when reading the store file.</p> <p>disabled—the server does not validate checksum data when reading the store file.</p>
Periodic Truncation	<p>enabled—the EMS server occasionally truncates the store file, relinquishing unused disk space.</p> <p>disabled—the EMS server does not truncate the store file to relinquish unused disk space.</p>
Destination Defrag Batch Size	The size of the batch used by the destination defrag feature.
File Size	The size of the store file, including unused allocated file space.
Free Space	The amount of unused allocated file space.
Fragmentation	The level of fragmentation in the file.
Used Space	The amount of used space in the file.
Message Size	Total size of all messages in the file.
Swapped Size	The total size of swapped messages in the file.
Disk Write Rate	The number of bytes written per second.
Headings specific to mstores	
Note that output for mstores includes many of the same fields available to file-based stores.	
Discard Scan Interval	The maximum length of time that the EMS server takes to examine all messages in the mstore. This interval is controlled with the <code>scan_iter_interval</code> parameter in the <code>stores.conf</code> file.
Discard Scan Interval Bytes	The bytes read and processed every Discard Scan Interval. This number is proportional to the mstore file size, and must be kept within the limits of your storage medium. See Understanding mstore Intervals on page 33 for more information.

Table 26 *show store — table Information*

Heading	Description
First Scan Finished	<p><code>true</code>—all the data in the store has been examined at least once since the EMS server startup.</p> <p><code>false</code>—not all data has been examined since the EMS server last started. When <code>false</code>, certain server statistics (such as the Message Count field) may be underreported as a result of expired or purged messages still in the store. See Implications for Statistics on page 35 for more information.</p>
Disk Write Rate	The number of bytes written per second.
Headings specific to database stores	
JDBC Driver Name	The name of the JDBC database server.
JDBC URL	The location of the JDBC database server.
Username	The username that the EMS server uses to access the database.
Dialect	The SQL dialect used to construct SQL commands.

show stores

```
show stores
```

Print a list of the server's stores.

show topic

```
show topic topic-name
```

Table 27 *show topic — table Information*

Heading	Description
Topic	Full name of the topic.
Type	<p><code>dynamic</code>—created by a client</p> <p><code>static</code>—configured by an administrator</p>

Table 27 *show topic — table Information (Cont'd)*

Heading	Description
Properties	A list of property names that are set on the topic, and their values. For an index list of property names, see Destination Properties on page 58 .
JNDI Names	A list of explicitly assigned JNDI names that refer to this topic.
Bridges	A list of bridges from this topic to other destinations.
Subscriptions	Number of subscriptions on this topic. (This count also includes durable subscriptions.)
Durable Subscriptions	The number of durable subscriptions on the topic.
Consumers	<p>Number of active consumers on this topic.</p> <p>Note: When a durable consumer is offline, it is not included in the count reported here.</p> <p>However, if this command is performed on an EMS 7.x or earlier server, the count also includes offline durable consumers.</p>
Durable Consumers	<p>Number of active durable consumers on this topic.</p> <p>Note: When a durable consumer is offline, it is not included in the count reported here.</p> <p>However, if this command is performed on an EMS 7.x or earlier server, the count also includes offline durable consumers.</p>
Pending Msgs	The total number of messages sent but not yet acknowledged by the consumer, followed by the number of persistent messages in parenthesis. These counts include copies sent to multiple subscribers.
Pending Msgs Size	Total size of all pending messages, followed by the size of all persistent messages in parenthesis.
The server accumulates the following statistics only when the administrator has enabled statistics. Otherwise these items are zero.	

Table 27 *show topic — table Information (Cont'd)*

Heading	Description
Total Inbound Msgs	Cumulative count of all messages delivered to the topic.
Total Inbound Bytes	Cumulative total of message size over all messages delivered to the topic.
Total Outbound Msgs	Cumulative count of messages consumed from the topic by consumers. Each consumer of a message increments this count independently of other consumers, so one inbound message results in <i>n</i> outbound messages (one per consumer).
Total Outbound Bytes	Cumulative total of message size over all messages consumed from the topic by consumers. Each consumer of a message contributes this total independently of other consumers.

show topics

```
show topics [pattern-name [notemp|static|dynamic]
[first=n|next=n|last=n]]
```

If a *pattern-name* is not entered, this command shows a list of all topics.

If a *pattern-name* is entered (for example `foo.*` or `foo.>`) this command shows a list of topics that match that pattern. See [Wildcards * and > on page 77](#) for more information about using wildcards.

You can further refine the list of topics that match the pattern by using one of the following parameters:

- `notemp` — do not show temporary topics
- `static` — show only static topics
- `dynamic` — show only dynamic topics

When a *pattern-name* is entered, you can also cursor through the list of topics using one of the following commands, where *n* is whole number:

- `first=n` — show the first *n* topics
- `next=n` — show the next *n* topics
- `last=n` — show the next *n* topics and terminate the cursor

The cursor examines *n* topics and displays topics that match the *pattern-name*. Because it does not traverse the full list of topics, the cursor may return zero or fewer than *n* topics. To find all matching topics, continue to use `next` until you receive a `Cursor complete` message.

The `show topics` command prints a table of information described in [Table 28](#).

Table 28 *show topics* — table information (Sheet 1 of 2)

Heading	Description
Topic Name	Name of the topic. If the name is prefixed with an asterisk (*), then the topic is temporary or was created dynamically. Properties of dynamic and temporary topics cannot be changed.
SNFGEIBCTM	<p>Prints information on the topic properties in the order:</p> <p>(S)ecure (N)sender_name or sender_name_enforced (F)ailsafe (G)lobal (E)xport (I)mport (B)ridge (C)flowControl (T)race (M)ulticast</p> <p>The characters in the value section show:</p> <ul style="list-style-type: none">- Property not present+ Property is present, and was set on the topic itself* Property is present, and was inherited from another topic <p>Note that inherited properties cannot be removed.</p>
Subs	<p>Number of current subscriptions on the topic, including durable subscriptions.</p> <p>If this command is performed on an EMS 7.x or earlier server, the count reflects the number of <i>subscribers</i>, not the number of subscriptions.</p>
Durs	<p>Number of durable subscriptions on the topic.</p> <p>If this command is performed on an EMS 7.x or earlier server, the count reflects the number of durable <i>subscribers</i>, not the number of subscriptions.</p>

Table 28 *show topics* — table information (Sheet 2 of 2)

Heading	Description
All Msgs	
Msgs	The total number of messages sent but not yet acknowledged by the consumer. This count includes copies sent to multiple subscribers. To see the count of actual messages (not multiplied by the number of topic subscribers) sent to all destinations, use the show server command.
Size	Total size of pending messages
Persistent Msgs	
Msgs	The total number of persistent messages sent but not yet acknowledged by the consumer.
Size	Total size of pending persistent messages

For more information, see [Destination Properties on page 58](#).

show subscriptions

```
show subscriptions [topic=name] [name=sub-name] [shared=only|none]
[durable=only|none] [sort=msgs|topic|name|cons|id]
```

This command prints information about all topic subscriptions, or only subscriptions matching specified filters. Command output is controlled using the `sort` parameter.

If `topic=name` is specified, then only subscriptions on destinations matching specified topic are shown. If `name=sub-name` is specified, then only subscriptions of that name are shown.

If `durable=only` is specified, then only durable subscriptions are shown.

If `durable=none` is specified, then only non-durable subscriptions are shown.

If `shared=only` is specified, then only shared subscriptions are shown.

If `shared=none` is specified, then only unshared subscriptions are shown.

The parameter `sort` allows you to specify how the command output is sorted in the output table. You can use to sort by by number of pending messages, topic name, subscription name, number of consumers on that subscription, or the subscription's identifier.

The `show subscriptions` command prints a table of information described in [Table 29](#).

Table 29 *show subscriptions* — table information

Heading	Description
Id	The ID of the subscription.
T	The subscription type: <ul style="list-style-type: none">• T — non-durable subscription• D — durable subscription
Topic	Name of the topic associated with the subscription.
Name	Name of the subscription (durable or shared name). If this is an unshared non-durable subscription, this value is empty.
SS	Description of columns: <ul style="list-style-type: none">• S - '+' if the subscription has a selector, '-' otherwise.• S - '+' if the subscription is shared, '-' otherwise.
Cons Count	The number of active consumers on this subscription. For an unshared non-durable subscription, the value is always 1. For a durable subscription, the value can be 0, meaning that there is no active consumer and the subscription is offline.
Pend Msgs	Total number of messages pending for the subscription.
Pend Size	Combined size of messages pending for the subscription. Value is rounded and shown in bytes, (K)ilobytes, (M)egabytes or (G)igabytes.
Uptime	The length of time, in hours, minutes, and seconds, since the subscription was created.

show transaction

```
show transaction XID
```

Shows a list of messages that were sent or received within the specified transaction. This command returns information on transactions in prepared, ended, and roll back states only. Transactions in a suspended or active state are not included.

[Table 30](#) describes the information shown in each column.

Table 30 show transactions — table information (Sheet 1 of 2)

Heading	Description
State	Transaction state: <ul style="list-style-type: none"> • A active • E ended • R rollback only • P prepared • S suspended Suspended transactions can be rolled back, but cannot be rolled forward (committed).
Remaining time before timeout	The seconds remaining before the TX timeout is reached. For example, 3 sec. This field is only applicable for transactions in State ENDSUCCESS or ROLLBACKONLY.
Messages to be consumed	
Message ID	The message ID of the message. null indicates the message ID could not be obtained or was disabled. If track_message_ids is not enabled, this field displays Disabled.
Type	The destination type to which the message was sent: <ul style="list-style-type: none"> • Q queue • T topic
Destination	The destination name to which the message was sent. null indicates that destination could not be found.
Consumer ID	The consumer ID of the Consumer that is consuming the message. Zero indicates that the consumer is offline.

Table 30 *show transactions* — table information (Sheet 2 of 2)

Heading	Description
Messages to be produced	
Message ID	The message ID of the message. null indicates the message ID could not be obtained or was disabled. If <code>track_message_ids</code> is not enabled, this field displays Disabled.
Type	The destination type to which the message was sent: <ul style="list-style-type: none">• Q queue• T topic
Destination	The destination name to which the message was sent. null indicates that destination could not be found.
JMSTimestamp	The timestamp indicating the time at which the message was created.

show transactions

`show transactions`

Shows the XID for all client transactions that were created using the XA or MS DTC interfaces. Each row presents information about one transaction. The XID is the concatenation of the Format ID, GTrid Len, Bqual Len, and Data fields for a transaction. For example, if `show transactions` returns the row:

State	Format	ID	GTrid	Len	Bqual	Len	Data
E	0		6		2		branchid

then the XID is 0 6 2 branchid. Note that the spaces are required.

[Table 31](#) describes the information shown in each column.

Table 31 show transactions — table information

Heading	Description
State	Transaction state: <ul style="list-style-type: none"> • A active • E ended • R rollback only • P prepared • S suspended Suspended transactions can be rolled back, but cannot be rolled forward (committed).
Format ID	The XA transaction format identifier. 0 = OSI CCR naming is used >0 = some other format is used -1 = NULL
GTrid Len	The number of bytes that constitute the global transaction ID.
Bqual Len	The number of bytes that constitute the branch qualifier.
Data	The global transaction identifier (gtrid) and the branch qualifier (bqual).

show transport

```
show transport transport
```

Displays the configuration for the specified transport defined in [transports.conf](#).

See [Configuring Transports for Rendezvous on page 404](#) and [Configuring Transports for SmartSockets on page 427](#) for details.

show transports

```
show transports
```

Lists all configured transport names in [transports.conf](#).

show user

```
show user user-name
```

Shows user name and description. If no user name is specified, this command displays the currently logged in user.

For users defined externally, there is an asterisk in front of the user name.

show users

```
show users
```

Shows all users.

For users defined externally, there is an asterisk in front of the user name. Only currently connected external users are shown.

showacl admin

```
showacl admin
```

Shows all administrative permissions for all users and groups, but does not include administrative permissions on destinations.

showacl group

```
showacl group group-name [admin]
```

Shows all permissions set for a given group. Shows the group and the set of permissions. You can optionally specify `admin` to show only the administrative permissions for destinations or principals. Specifying `showacl admin` shows all administrative permissions for all users and groups (not including administrative permissions on destinations).

showacl queue

```
showacl queue queue-name [admin]
```

Shows all permissions set for a queue. Lists all entries from the `acl` file. Each entry shows the “grantee” (user or group) and the set of permissions. You can optionally specify `admin` to show only the administrative permissions for destinations or principals. Specifying `showacl admin` shows all administrative permissions for all users and groups (not including administrative permissions on destinations).

showacl topic

```
showacl topic topic-name [admin]
```

Shows all permissions set for a topic. Lists all entries from the acl file. Each entry shows the “grantee” (user or group) and the set of permissions. You can optionally specify `admin` to show only the administrative permissions for destinations or principals. Specifying `showacl admin` shows all administrative permissions for all users and groups (not including administrative permissions on destinations).

showacl user

```
showacl user user-name [admin | all | admin-all]
```

Shows the user and the set of permissions granted to the user for destinations and principals.

`showacl user username` — displays permissions granted directly to the user. (An administrator can use this form of the command to view own permissions, even without permissions to view any other user permissions.)

`showacl user username admin` — displays administrative permissions granted directly to the user.

`showacl user username all` — displays direct and inherited (from groups to which the user belongs) permissions.

`showacl user username admin-all` — displays all administrative permissions for a given user (direct and inherited)



The output from this command displays inherited permissions prefixed with a '*'. Inherited permissions cannot be changed. An attempt to revoke an inherited permission for the principal user will not change the permission.

shutdown

```
shutdown
```

Shuts down currently connected server.

suspend route

```
suspend route route-name
```

Suspends outgoing messages to the named route.

Message flow can be recovered later using the command `resume route`.

time

```
time [on | off]
```

Specifying `on` places a timestamp before each command's output. By default, the timestamp is `off`.

timeout

```
timeout [seconds]
```

Show or change the current command timeout value. The timeout value is the number of seconds the Administration Tool will wait for a response from the server after sending a command.

By default, the timeout is 30 seconds. When `timeout` is entered with the optional *seconds* parameter, the timeout value is reset to the specified number of seconds. When entered without parameter, the current timeout value is returned.

transaction commit

```
transaction commit XID
```

Commits the transaction identified by the transaction ID. The transaction must be in the ended or prepared state. To obtain a transaction ID, issue the `show transactions` command, and cut and paste the XID into this command.

transaction rollback

```
transaction rollback XID
```

Rolls back the transaction identified by the transaction ID. The transaction must be in the ended, `rollback only`, or the prepared state. To obtain a transaction ID, issue the `show transactions` command, and cut and paste the XID into this command.



Messages sent to a queue with `prefetch=none` and `maxRedelivery=number` properties are not received *number* times by an EMS application that receives in a loop and does an XA rollback after the XA prepare phase.

updatecrl

```
updatecrl
```

Immediately update the server's certificate revocation list (CRL).

whoami

whoami

Alias for the show user command to display the currently logged in user.

Chapter 7 **Using the Configuration Files**

This chapter describes configuring TIBCO Enterprise Message Service.

Topics

- [Location of Configuration Files, page 186](#)
- [Mechanics of Configuration, page 186](#)
- [tibemsd.conf, page 187](#)
- [Using Other Configuration Files, page 238](#)

Location of Configuration Files

The installation process places configuration files in two directories:

- *config-file-directory/cfmgmt/ems/data/* contains a subset of configuration files suitable for quickly testing the installation. The *config-file-directory* is specified during the Configuration Directory step installation process.
- *EMS_HOME/samples/config/* contains the more complete set of sample configuration files. For deployment, we recommend copying files from this directory to a production configuration directory, and modifying those copies.

When selecting a production configuration directory, we recommend using a file system with regular backup commensurate with your need for reliability and disaster recovery. It is essential that the EMS server have both read and write privileges in the configuration directory.

Mechanics of Configuration

Configuration Files	The EMS server reads configuration files only once, when the server starts. It ignores subsequent changes to the configuration files. If you change a configuration file, use the shutdown command from the EMS Administration Tool to shutdown the server and then restart the server as described in Running the EMS Server on page 107 .
---------------------	---

Administrative Requests	You can also change the server configuration with administrative requests, using either <code>tibemsadmin</code> (a command line tool), the Java or .NET administrative APIs, or TIBCO Administrator™ (a separate TIBCO product).
-------------------------	---

When the server validates and accepts an administrative request, it writes the change to the appropriate configuration file as well (overwriting any manual changes to that file). This policy keeps configuration files current in case the server restarts (for example, in a fault-tolerant situation, or after a hardware failure).

Re-installing or updating EMS overwrites the files in the `bin/` and `samples/config/` directories. Do not use these directories to configure your deployment.

tibemsd.conf

The main configuration file controls the characteristics of the EMS server. This file is usually named `tibemsd.conf`, but you can specify another file name when starting the server. You can find more information about starting the server in [Running the EMS Server on page 107](#).

An example of the `tibemsd.conf` file is included in the `config-file-directory/cfmgmt/ems/data/` directory, where `config-file-directory` is specified during TIBCO Enterprise Message Service installation. You can edit this configuration file with a text editor. There are a few configuration items in this file that can be altered using the administration tool, but most configuration parameters must be set by editing the file (that is, the server does not accept changes to those parameters). See [Chapter 6, Using the EMS Administration Tool, on page 123](#) for more information about using the administration tool.

Several parameters accept boolean values. In the description of the parameter, one specific set of values is given (for example, `enable` and `disable`), but all parameters that accept booleans can have the following values:

- `enable`, `enabled`, `true`, `yes`, `on`
- `disable`, `disabled`, `false`, `no`, `off`

Parameters that take multiple elements cannot contain spaces between the elements, unless the elements are enclosed in starting and ending double quotes. Parameters are limited to line lengths no greater than 256,000 characters in length.

The following table summarizes the parameters in `tibemsd.conf` according to category. The sections that follow provide more detail on each parameter.

Table 32 *tibemsd.conf* Parameters

Parameter	Description	See Page
Global System Parameters		
authorization	Enable or disable server authorization.	199
compliant_queue_ack	Guarantees that a message will not be redelivered after a client has successfully acknowledged its receipt from a routed queue.	199
disconnect_non_acking_consumers	Causes the server to review unacknowledged pending messages size and counts in consumers.	199

Table 32 *tibemsd.conf* Parameters

Parameter	Description	See Page
<code>flow_control</code>	Enable or disable flow control for destinations.	200
<code>listen</code>	Specifies the port on which the server is to listen for connections from clients.	201
<code>max_msg_field_print_size</code>	Limits the size of string fields in tracing messages.	201
<code>max_msg_print_size</code>	Limits the size of the printed message of traced messages.	201
<code>npsend_check_mode</code>	Specifies when the server is to provide confirmation upon receiving a <code>NON_PERSISTENT</code> message from a producer.	202
<code>password</code>	Password used to authenticate with other routed servers that have <code>authorization</code> enabled.	202
<code>processor_ids</code>	Specifies the processors to be used for network I/O traffic.	203
<code>routing</code>	Enable or disable routing functionality for this server.	204
<code>selector_logical_operator_limit</code>	Limits the number of operators that the server reviews during selector evaluation.	204
<code>server</code>	Name of server.	204
<code>startup_abort_list</code>	Specifies conditions under which the server is to exit during its initialization sequence.	205
<code>user_auth</code>	Specifies the source of authentication information used to authenticate users attempting to access the EMS server.	206
<code>xa_default_timeout</code>	Specifies the TX timeout for XA transactions.	206

Table 32 *tibemsd.conf* Parameters

Parameter	Description	See Page
Storage File Parameter		
<code>store</code>	Specifies the directory in which the server stores data.	206
Connection and Memory Parameters		
<code>destination_backlog_swapout</code>	Specifies the maximum number of messages per destination that are stored in the server before message swapping is enabled.	207
<code>handshake_timeout</code>	Specifies the amount of time that the EMS server waits for an SSL connection to complete.	207
<code>max_client_msg_size</code>	Sets a maximum size for incoming messages.	207
<code>max_connections</code>	Specifies the maximum number of simultaneous client connections to the server.	207
<code>max_msg_memory</code>	Specifies the maximum memory the server can use for messages.	208
<code>msg_pool_block_size</code>	Specifies the size of the pool to be pre-allocated by the server to store messages.	208
<code>msg_swapping</code>	Enable or disable message swapping.	209
<code>reserve_memory</code>	Specifies the amount of memory to reserve for use in emergency situations.	209
<code>socket_send_buffer_size</code>	Sets the size of the send buffer used by clients when connecting to the EMS server.	210
<code>socket_receive_buffer_size</code>	Sets the size of the receive buffer used by clients when connecting to the EMS server.	210

Table 32 *tibemsd.conf* Parameters

Parameter	Description	See Page
Detecting Network Connection Failure Parameters		
<code>client_heartbeat_server</code>	Specifies the interval clients are to send heartbeats to the server.	211
<code>clock_sync_interval</code>	Periodically sends the EMS server's UTC time to clients.	211
<code>server_timeout_client_connection</code>	Specifies the period of time server will wait for a client heartbeat before terminating the client connection.	211
<code>server_heartbeat_server</code>	Specifies the interval this server is to send heartbeats to another server.	212
<code>server_timeout_server_connection</code>	Specifies the period of time this server will wait for a heartbeat from another server before terminating the connection to that server.	212
<code>server_heartbeat_client</code>	Specifies the interval this server is to send heartbeats to all of its clients.	212
<code>client_timeout_server_connection</code>	Specifies the period of time a client will wait for a heartbeat from the server before terminating the connection.	213
Fault Tolerance Parameters		
<code>ft_active</code>	Specifies the URL of the active server.	213
<code>ft_heartbeat</code>	Specifies the interval the active server is to send a heartbeat signal to the backup server to indicate that it is still operating.	213
<code>ft_activation</code>	Specifies the maximum length of time between heartbeat signals the backup server is to wait before assuming the active server has failed.	214

Table 32 *tibemsd.conf* Parameters

Parameter	Description	See Page
ft_reconnect_timeout	Specifies the maximum length of time the backup server is to wait for clients to reconnect after assuming the role of primary server in a failover situation.	214
ft_ssl_auth_only	Specifies whether the server allows a fault tolerant server to request the use of SSL only for authentication.	214
ft_ssl_identity	Specifies the server's digital certificate.	214
ft_ssl_issuer	Specifies the certificate chain member for the server.	215
ft_ssl_private_key	Specifies the server's private key.	215
ft_ssl_password	Specifies the password for private keys.	215
ft_ssl_trusted	Specifies the list of trusted certificates.	215
ft_ssl_rand_egd	Specifies the path for the installed entropy gathering daemon (EGD).	216
ft_ssl_verify_host	Specifies whether the fault-tolerant server should verify the other server's certificate.	216
ft_ssl_verify_hostname	Specifies whether the fault-tolerant server should verify the name in the CN field of the other server's certificate.	216
ft_ssl_expected_hostname	Specifies the name the server is expected to have in the CN field of the fault-tolerant server's certificate.	216
ft_ssl_ciphers	Specifies the cipher suites used by the server.	217
Message Tracking Parameters		
track_message_ids	Enable or disable message tracking by message ID.	217

Table 32 *tibemsd.conf* Parameters

Parameter	Description	See Page
<code>track_correlation_ids</code>	Enable or disable message tracking by correlation ID.	217
Multicast Parameters		
<code>multicast</code>	Enables or disables multicast in the EMS server.	217
<code>multicast_channels</code>	Specifies the configuration file where multicast channels are defined.	218
<code>multicast_daemon_default</code>	Specifies the default port on which the multicast daemon listens for connections from EMS clients.	218
<code>multicast_statistics_interval</code>	Specifies how often, in seconds, multicast statistics are generated for each channel.	218
TIBCO Rendezvous Parameters		
<code>tibrv_transports</code>	Enable or disable the TIBCO Rendezvous transports defined in <code>transports.conf</code> file.	219
TIBCO SmartSockets Parameters		
<code>module_path</code>	Specify the directory containing the TIBCO SmartSockets library files.	219
<code>tibss_transports</code>	Enable or disable the TIBCO SmartSockets transports defined in <code>transports.conf</code> file.	219
<code>tibss_config_dir</code>	Specifies the directory for SmartSockets configuration and message files.	220
Tracing and Log File Parameters		
<code>logfile</code>	Name and location of the server log file.	220
<code>log_trace</code>	Specifies the trace options on the file defined by the <code>logfile</code> parameter.	220

Table 32 *tibemsd.conf* Parameters

Parameter	Description	See Page
<code>logfile_max_count</code>	Specifies the maximum number of log files to be kept.	221
<code>logfile_max_size</code>	Specifies the maximum log file size before the log file is copied to a backup and then emptied.	221
<code>console_trace</code>	Specifies the trace options for output to <code>stderr</code> .	221
<code>client_trace</code>	Enable or disable client generation of trace output for opening or closing a connection, message activity, and transaction activity.	222
<code>trace_client_host</code>	Specifies whether the trace statements related to connections identify the host by its hostname, its IP address, or both.	222
Statistic Gathering Parameters		
<code>server_rate_interval</code>	Specifies the interval at which overall server statistics are averaged.	222
<code>statistics</code>	Enables or disables statistic gathering for producers, consumers, destinations, and routes.	223
<code>rate_interval</code>	Specifies the interval at which statistics for routes, destinations, producers, and consumers are averaged.	223
<code>detailed_statistics</code>	Specifies which objects should have detailed statistic tracking.	223
<code>statistics_cleanup_interval</code>	Specifies how long the server should keep detailed statistics if the destination has no activity.	224
<code>max_stat_memory</code>	Specifies the maximum amount of memory to use for detailed statistic gathering.	224

Table 32 *tibemsd.conf* Parameters

Parameter	Description	See Page
SSL Server Parameters		
<code>ssl_dh_size</code>	Specifies the size of the Diffie-Hellman key.	224
<code>ssl_server_ciphers</code>	Specifies the cipher suites used by the server.	224
<code>ssl_require_client_cert</code>	Specifies if the server is to only accept SSL connections from clients that have digital certificates.	225
<code>ssl_use_cert_username</code>	Specifies if a client's user name is to always be extracted from the CN field of the client's digital certificate.	225
<code>ssl_cert_user_specname</code>	Specifies a special username to identify which clients are to have their usernames taken from their digital certificates.	225
<code>ssl_server_identity</code>	Specifies the server's digital certificate.	226
<code>ssl_server_key</code>	Specifies the server's private key.	226
<code>ssl_password</code>	Specifies the password for private keys.	226
<code>ssl_server_issuer</code>	Specifies the certificate chain member for the server.	227
<code>ssl_server_trusted</code>	Specifies the list of CA root certificates the server trusts as issuers of client certificates.	227
<code>ssl_rand_egd</code>	Specifies the path for the installed entropy gathering daemon (EGD).	227
<code>ssl_crl_path</code>	Specifies the pathname to the certificate revocation list (CRL) files.	228
<code>ssl_crl_update_interval</code>	Specifies the interval at which the server is to update its CRLs.	228

Table 32 *tibemsd.conf* Parameters

Parameter	Description	See Page
ssl_auth_only	Specifies whether the server allows clients to request the use of SSL only for authentication.	228
fips140-2	Enables the server for FIPS compliance.	228
LDAP Parameters		
ldap_url	Specifies the URL of the external directory server.	228
ldap_principal	Specifies the distinguished name (DN) of the LDAP administrator.	229
ldap_credential	Specifies the password associated with the user defined in the <code>ldap_principal</code> property.	229
ldap_cache_enabled	Enables or disables caching of LDAP data.	229
ldap_cache_ttl	Specifies the maximum time that cached LDAP data is retained before it is refreshed.	229
ldap_conn_type	Specifies the type of connection that the server uses to get LDAP information.	229
ldap_tls_cacert_file	Specifies the file that contains the CA certificate the EMS server trusts to sign the LDAP server's certificate.	230
ldap_tls_cacert_dir	When there are two or more CA certificates in the verify chain, use this parameter to specify the directory containing the CA certificates.	230
ldap_tls_cipher_suite	Specifies the cipher suite to use for encryption on secure LDAP connections.	230
ldap_tls_rand_file	Specifies the file containing random data for encryption.	230

Table 32 *tibemsd.conf* Parameters

Parameter	Description	See Page
<code>ldap_tls_cert_file</code>	Specifies the file containing the certificate that identifies the EMS server to the LDAP server.	231
<code>ldap_tls_key_file</code>	Specifies the file containing the private key required by the LDAP server to authenticate the client.	231
<code>ldap_user_class</code>	Specifies the name of the LDAP object class that stores users.	231
<code>ldap_user_attribute</code>	Specifies the name of the attribute on the user object class that holds the name of the user.	231
<code>ldap_user_base_dn</code>	Specifies the base distinguished name (DN) of the LDAP tree that contains the users.	231
<code>ldap_user_scope</code>	Specifies how deeply under the base DN to search for users.	232
<code>ldap_user_filter</code>	Specifies the LDAP search filter for finding a given user name.	232
<code>ldap_all_users_filter</code>	Specifies the LDAP search filter for finding all users beneath the user base DN.	232
<code>ldap_group_base_dn</code>	Specifies the base distinguished name (DN) of the LDAP tree that contains groups.	232
<code>ldap_group_scope</code>	Specifies how deeply under the base DN to search for groups.	232
<code>ldap_group_filter</code>	Specifies the LDAP search filter for finding a group with a given group name.	233
<code>ldap_all_groups_filter</code>	Specifies the LDAP search filter for finding all groups beneath the group base DN.	233
<code>ldap_static_group_class</code>	Specifies the name of the LDAP object class that stores static groups.	233

Table 32 *tibemsd.conf* Parameters

Parameter	Description	See Page
ldap_static_group_attribute	Specifies the name of the attribute on the static group object class that holds the name of the group.	233
ldap_static_group_member_filter	Specifies the LDAP serch filter for finding all static members of a group	234
ldap_static_member_attribute	Specifies the attribute of an LDAP static group object that specifies the distinguished names (DNs) of the members of the group.	234
ldap_dynamic_group_class	Specifies the name of the LDAP object class that stores dynamic groups.	234
ldap_dynamic_group_attribute	Specifies the name of the attribute on the dynamic group object class that holds the name of the group.	234
ldap_dynamic_member_url_attribute	Specifies the attribute of the dynamic LDAP group object that specifies the URLs of the members of the dynamic group.	235
Extensible Security Parameters		
jaas_classpath	Includes the JAR files and dependent classes used by the JAAS LoginModule.	235
jaas_config_file	Specifies the location of the JAAS configuration file used to run a custom authentication LoginModule.	235
jaas_login_timeout	Specifies the length of time, in milliseconds, that the server waits for the JAAS authentication module to execute and respond.	235
jaci_classpath	Includes the JAR files and dependent classes used by the JACI custom permissions module.	236

Table 32 *tibemsd.conf* Parameters

Parameter	Description	See Page
<code>jaci_class</code>	Specifies the name of the class that implements the extensible permissions interface.	236
<code>jaci_timeout</code>	Specifies the length of time, in milliseconds, that the server waits for the JACI permissions module to execute and respond.	236
JVM Parameters		
<code>jre_library</code>	Enables the JVM in the EMS server.	237
<code>jre_option</code>	Passes command line options to the JVM at start-up.	237

Global System Parameters

authorization

`authorization = enabled | disabled`

Enable or disable server authorization.

Authorization is disabled by default. If you require that the server verify user credentials and permissions on secure destinations, you must enable this parameter.

See [Enabling Access Control on page 275](#) for more information.

For example:

`authorization = enabled`

See [Chapter 8, Authentication and Permissions, on page 265](#) for more information about these parameters.

compliant_queue_ack

`compliant_queue_ack = enable | disable`

Guarantees that, once a client successfully acknowledges a message received from a routed queue, the message will not be redelivered. This is accomplished by the EMS server waiting until the message has been successfully acknowledged by the queue's home EMS server before sending the response to the client.

The `compliant_queue_ack` parameter is enabled by default. Because of the extra overhead incurred with compliant queue acknowledgments, you can disable this feature when performance is an issue. If compliant queue acknowledgement is disabled and a message is redelivered, the message's `JMSRedelivered` indicator will be set.

disconnect_non_acking_consumers

`disconnect_non_acking_consumers = enabled | disabled`

This parameter works in conjunction with the `maxbytes` and `maxmsgs` destination properties. In situations where consumers consume messages but do not acknowledge them, the messages are held in the server until they are confirmed. This can push the server above the set limits.

When enabled, `disconnect_non_acking_consumers` causes the server to check the number and size of pending messages sent to a consumer. If the `maxbytes` or `maxmsgs` limit is reached and the consumer has not acknowledged its messages, the server discards the messages sent to the consumer and disconnects the consumer's connection. This protects the server against applications that consume messages without ever acknowledging them.

Before enabling this property, ensure that the `maxbytes` and `maxmsgs` limits are set with reference to the [prefetch](#) setting, the size of the transaction (if transacted receive), or number of messages acknowledged when using client or explicit client acknowledgment mode. Otherwise the server may disconnect the consumer before it has a chance to acknowledge the messages.

When routes are deployed, all routed servers should use the same `disconnect_non_acking_consumers` setting. Additionally, if `maxbytes` or `maxmsgs` is set for a global destination, the same setting should be applied on all servers. The server does not discard or disconnect a routed consumer, since disconnecting the route may impact other well-behaved applications. Servers discard and disconnect their local consumers, which other servers involved are made aware of and discard messages for those remote consumers accordingly.

This parameter is disabled by default.

flow_control

```
flow_control = enable | disable
```

Specifies whether flow control for destinations is enabled or disabled.

By default, flow control is disabled. When flow control is enabled, the `flowControl` property on each destination specifies the target maximum storage for pending messages on the destination.

See [Flow Control on page 87](#) for more information about flow control.

listen

```
listen=protocol://servername:port
```

Specifies the port on which the server is to listen for connections from clients.

For example:

```
listen=tcp://localhost:7222
```

If you are enabling SSL, for example:

```
listen=ssl://localhost:7222
```

You can use multiple `listen` entries if you have computers with multiple interfaces. For example:

```
listen=tcp://localhost:7222
```

```
listen=tcp://localhost:7224
```

If `localhost` is specified, or if the *servername* is not present, then the server uses every available interface. For example:

```
listen=tcp://7222
```

```
listen=ssl://7243
```

You can use an IP address instead of a host name. For example:

```
listen=tcp://192.168.10.107:7222
```

When specifying an IPv6 address, use square brackets around the address specification. For example:

```
listen=tcp://[2001:cafe::107]:7222
```

max_msg_field_print_size

```
max_msg_field_print_size = size [KB|MB|GB]
```

Limits the size of string fields in tracing messages. If a string field is larger than *size*, the field is truncated in the tracing message.

Specify signed 32-bit integer values as KB, MB or GB. The minimum permitted size is 1 KB. By default, the field limit is 1 KB.

max_msg_print_size

```
max_msg_print_size = size [KB|MB|GB]
```

Limits the size of the printed message of traced messages. If the message is larger than *size*, the message is truncated.

Specify signed 32-bit integer values as KB, MB or GB. The minimum permitted size is 8 KB. By default, the field limit is 8 KB.

npsend_check_mode

`npsend_check_mode = [always | never | temp_dest | auth | temp_auth]`

Specifies when the server is to provide confirmation upon receiving a `NON_PERSISTENT` message from a producer.

The `npsend_check_mode` parameter applies only to producers sending messages using `NON_PERSISTENT` delivery mode and non-transactional sessions.

Message confirmation has a great deal of impact on performance and should only be enabled when necessary. The circumstances in which a producer might want the server to send confirmation a `NON_PERSISTENT` message are:

- When `authorization` is enabled, so the producer can take action if permission to send the message is denied by the server.
- When sending to a temporary destination, so the producer can take action if the message is sent to a temporary destination that has been destroyed.
- The message exceeded queue/topic limit (requires `rejectIncoming` policy for topics).
- Bridging of the message has failed.
- The server is out of memory or has encountered some other severe error.

The possible `npsend_check_mode` parameter modes are:

- `default` (no mode specified) - same behavior as in EMS 4.3 and prior. This means the server only provides confirmation of a `NON_PERSISTENT` message if `authorization` is enabled.
- `always` - the server always provides confirmation of a `NON_PERSISTENT` message.
- `never` - the server never provides confirmation of `NON_PERSISTENT` messages.
- `temp_dest` - the server provides confirmation of a `NON_PERSISTENT` message only when sending to a temporary destination.
- `auth` - the server provides confirmation of a `NON_PERSISTENT` message only if `authorization` was enabled when the connection was created.
- `temp_auth` - the server provides confirmation of a `NON_PERSISTENT` message if sending to a temporary destination or if `authorization` was enabled when the connection was created.

password

`password = password`

Password used to log in to other routed servers that have `authorization` enabled. See [Routing and Authorization on page 531](#) for details.

processor_ids

```
processor_ids = processor-id1, processor-id2, ...
```

Setting this parameter causes the EMS Server to start as many network I/O threads as there are processor IDs specified in the list. Each network I/O thread is bound to the given processor ID, which means that the thread can execute only on that processor.



Do not use this parameter if the default behavior provides sufficient throughput.

Specify the *processor-id* as an integer, starting at 0 and continuing to the number of processors available, minus 1. For example, if you have four processors, the valid processor IDs are 0, 1, 2, and 3. Note that the IDs can be listed in any order.

On startup, the parameter is parsed and the server refuses to start (regardless of the presence of the `startup_abort_list` parameter) if:

1. The list is malformed. That is, if it contains invalid values such as non-numeric elements.
2. A processor ID specified is greater than the number of processors available on the machine. For example, if the processor ID 4 is specified in a machine with only 4 processors. (Valid IDs for a 4-processor machine are 0, 1, 2 and 3.)
3. The server is unable to bind a network I/O thread to a given processor ID. This can happen when the processor ID has been disabled, or the `tibemspd` process has been restricted by the system administrator to a set of processors that does not contain this processor ID.



Do not use hyper threading.

For instance, consider a machine with 24 processors, with 2 dies and processor IDs ranging from 0 to 5 and 12 to 17 on the first die, and 6 to 11 and 18 to 23 on the second die. In this example, you should specify processor IDs in either the 0 to 5 range, or the 6 to 11 range.

Specifying processor IDs 0 and 12 in the list would cause thrashing because two network I/O threads would be bound to the same processor (or core). Also, for optimal performance, processor IDs should be from the same die.

This parameter can be used in conjunction with the `stores.conf` parameter `processor_id`. For more information, see [Performance Tuning on page 121](#).

routing

```
routing = enabled | disable
```

Enables or disables routing functionality for this server.

For example:

```
routing = enabled
```

See [Chapter 20, Working With Routes, on page 509](#) for more information about routing.

selector_logical_operator_limit

```
selector_logical_operator_limit = number
```

Limit the number of operators that the server reviews during selector evaluation.

The server evaluates operators until reaching the specified *number* of false conditions. The server then stops evaluating further to protect itself from too many recursive evaluations. A very long selector clause, such as one including many OR conditions, can cause recursive selector evaluation and lead to a stack overflow in the EMS server.

number may be any positive integer. The default value is 5000. Zero is a special value, indicating no limit.

For example, if `selector_logical_operator_limit = 10` and the selector is:

```
a=1 or b=2 or c=3 or d=4 or e=5 or f=6 or g=7 or h=8 or i=9 or j=10
or k=11 or l=12 or m=13 or n=14 or o=15 or p=16 or q=17 or r=18 or
s=19 or t=20 or u=21 or v=22 or w=23 or x=24 or y=25 or z=26
```

if the first 10 conditions are false, the server stops further evaluation.

server

```
server = serverName
```

Name of server.

Server names are limited to at most 64 characters, and may not include the dot character (.).

startup_abort_list

```
startup_abort_list=[SSL,TRANSPORTS,CONFIG_FILES,CONFIG_ERRORS,
DB_FILES,MULTICAST]
```

Specifies conditions that cause the server to exit during its initialization sequence.

You may specify any subset of the conditions in a comma-separated list. The list cannot contain spaces between the elements, unless the elements are enclosed in starting and ending double quotes. If a space is included but not enclosed in quotation marks, the server ignores any conditions following the space.

Conditions that do not appear in the list are ignored by the server. The default is an empty list.

The conditions are:

- **SSL**—If SSL initialization fails, then it exits.
- **TRANSPORTS**—If any of the transports cannot be created as specified in the configuration files, then it exits.
- **CONFIG_FILES**—If any configuration file listed in `tibemsd.conf` does not exist, then it exits.
- **CONFIG_ERRORS**—If the server detects any errors while reading the config files, then it exits.

Note that the `tibemsd` silently ignores any unknown parameters when it is started using the JSON configuration. For example, no configuration errors are thrown if the `tibemsd.json` file contains an obsolete parameter.

- **DB_FILES**—If the server cannot find one or more of its stores, then it exits. Stores include the default store files as well as any file or database stores configured in the [stores.conf](#) configuration file.

Note that if `DB_FILES` is *not* included in the `startup_abort_list` and the server cannot find a store, the server will create the missing file or database. For best results, do not include `DB_FILES` the first time a server is started, allowing it to create the files. After after initial startup or a major store configuration change (such as the addition of a new store), include `DB_FILES` in the list so that on restart the server will only start if all the configured files are present.

- **MULTICAST**—If the server detects that it cannot send multicast messages, then it exits.

Note that if `MULTICAST` is *not* in the `startup_abort_list` and multicast initialization fails, applications creating consumers on multicast-enabled topics will receive messages over TCP. This is important to consider if your network cannot handle the bandwidth allocated for multicast when it is sent over a TCP connection.

user_auth

```
user_auth = [local, ldap, jaas]
```

Specifies the source of user authentication information.

This parameter can have one or more of the following values (separated by comma characters):

- `local`—obtain user authentication information from the local EMS server user configuration.
- `ldap`—obtain user authentication information from an LDAP directory server (see the LDAP-specific configuration parameters).
- `jaas`—obtain user authentication information from a custom authentication module (see [Extensible Authentication on page 292](#)).

Each time a user attempts to authenticate, the server seeks corresponding authentication information from each of the specified locations in the order that this parameter specifies. The EMS server accepts successful authentication using any of the specified sources.



The `user_auth` setting does not affect authentication of the default administrator. The server always authenticates the `admin` user from the local configuration file. See [Assign a Password to the Administrator on page 126](#) for more information.

xa_default_timeout

```
xa_default_timeout = seconds
```

Specifies the default TX timeout, in seconds, for XA transactions. The default is 0, which specifies no timeout.

The default timeout setting cannot be changed dynamically. However, you can specify a different transaction timeout for each individual XA resource using the API.

Storage File Parameters

The parameter described here configures file-based and mstores. For information about database stores, see [Chapter 10, Using Database Stores](#).

store

```
store = directory
```

Directory in which the server stores data files. For example:

```
store = /usr/tmp
```


Connection and Memory Parameters

destination_backlog_swapout

`destination_backlog_swapout = number`

Specifies the number of messages that may be stored in the server's memory before message swapping is enabled. The limit given is for each destination. For example, if the limit is 10,000 and you have three queues, the server can store up to 30,000 unswapped messages in memory.

The specified *number* may be any positive value. When `destination_backlog_swapout` is 0, the server attempts to immediately swap out the message.

By default, the limit for each destination is 1024 messages.

handshake_timeout

`handshake_timeout = seconds`

The amount of time (in seconds) that the EMS server waits for an SSL connection to complete. *seconds* may be any positive integer. The default value is 3 seconds.

When the timeout is reached, the EMS server closes the SSL connection and continues servicing other clients.

max_client_msg_size

`max_client_msg_size = size [KB|MB|GB]`

Maximum size allowed for an incoming message.

This parameter setting instructs the server to reject incoming messages that are larger than the specified size limit.

Specify whole numbers as KB, MB or GB. The maximum value is 2 GB.

When omitted or zero, the EMS server accepts and attempts to process messages of any size.

max_connections

`max_connections = number`

Maximum number of simultaneous client connections.

Set to 0 to allow unlimited simultaneous connections.

max_msg_memory

```
max_msg_memory = size [KB|MB|GB]
```

Maximum memory the server can use for messages.

This parameter lets you limit the memory that the server uses for messages, so server memory usage cannot grow beyond the system's memory capacity.

When `msg_swapping` is enabled, and messages overflow this limit, the server begins to swap messages from process memory to disk. Swapping allows the server to free process memory for incoming messages, and to process message volume in excess of this limit.

When the server swaps a message to disk, a small record of the swapped message remains in memory. If all messages are swapped out to disk, and their remains still exceed this memory limit, then the server has no room for new incoming messages. The server stops accepting new messages, and send calls in message producers result in an error. (This situation probably indicates either a very low value for this parameter, or a very high message volume.)

Specify units as KB, MB or GB. The minimum value is 8 MB. The default value of 0 (zero) indicates no limit.

For example:

```
max_msg_memory = 512MB
```

msg_pool_block_size

```
msg_pool_block_size size
```



Consult with your TIBCO support representative before using this parameter.

To lessen the overhead costs associated with `malloc` and `free`, the server pre-allocates pools of storage for messages. This parameter determines the behavior of these pools. Performance varies depending on operating system platform and usage patterns.

The *size* argument determines the approximate number of internal message structs that a block or pool can accommodate (not the number of bytes).

`msg_pool_block_size` instructs the server to allocate an *expandable* pool. Each time the server exhausts the pool, the server increases the pool by this size, as long as additional storage is available. The value may be in the range 32 to 65536.

When this parameter is not present, the default is `msg_pool_block_size 128`.

msg_swapping

```
msg_swapping = enable | disable
```

This parameter enables and disables the message swapping feature (described above for [max_msg_memory](#)).

The default value is enabled, unless you explicitly set it to disabled.

reserve_memory

```
reserve_memory = size
```

When `reserve_memory` is non-zero, the daemon allocates a block of memory for use in emergency situations to prevent the EMS server from being unstable in low memory situations. When the daemon process exhausts memory resources, it disables clients and routes from producing new messages, and frees this block of memory to allow consumers to continue operation (which tends to free memory).

The EMS server attempts to reallocate its reserve memory once the number of pending messages in the server has dropped to 10% of the number of pending messages that were in the server when it experienced the allocation error. If the server successfully reallocates memory, it begins accepting new messages.

The `reserve_memory` parameter only triggers when the EMS server has run out of memory and therefore is a reactive mechanism. The appropriate administrative action when an EMS server has triggered release of reserve memory is to drain the majority of the messages by consuming them and then to stop and restart the EMS server. This allows the operating system to reclaim all the virtual memory resources that have been consumed by the EMS server. A trace option, [MEMORY](#), is also available to help show what the server is doing during the period when it is not accepting messages.

Specify *size* in units of MB. When non-zero, the minimum block is 16MB. When absent, the default is zero.



There are a variety of limits that the user can set to prevent the EMS server from storing excessive messages, which can lead to situations where the EMS server runs out of memory. These include global parameters, such as [max_msg_memory](#), as well as destination properties such as [maxbytes](#). These limits should be used to prevent the `reserve_memory` mechanism from triggering.

socket_send_buffer_size

```
socket_send_buffer_size = size [KB|MB|GB]
```

Sets the size (in bytes) of the send buffer used by clients when connecting to the EMS server.

The specified *size* may be:

- any number greater than 512
- 0 to use the default buffer size
- -1 to skip the call for the specified buffer
- Optionally, specify units of KB, MB, or GB for units. If no units are specified, the file size is assumed to be in bytes.

When omitted or zero, the default buffer size is used.



On Linux platforms, omitting the parameter or specifying zero causes the EMS server to skip the value. In this case, Linux auto-tuning controls buffering.

socket_receive_buffer_size

```
socket_receive_buffer_size = size [KB|MB|GB]
```

Sets the size (in bytes) of the receive buffer used by clients when connecting to the EMS server.

The specified *size* may be:

- any number greater than 512
- 0 to use the default buffer size
- -1 to skip the call for the specified buffer
- Optionally, specify units of KB, MB, or GB for units. If no units are specified, the file size is assumed to be in bytes.

When omitted or zero, the default buffer size is used.



On Linux platforms, omitting the parameter or specifying zero causes the EMS server to skip the value. In this case, Linux auto-tuning controls buffering.

Detecting Network Connection Failure Parameters

This feature lets servers and clients detect network connection failures quickly. When these parameters are absent, or this feature is disabled, `tibemsd` closes a connection only upon the operating system notification.

client_heartbeat_server

`client_heartbeat_server = interval`

In a server-to-client connection, clients send heartbeats to the server at this interval (in seconds).

The `client_heartbeat_server` parameter must be specified when a `server_timeout_client_connection` is set. The `client_heartbeat_server` interval should be no greater than one third of the `server_timeout_client_connection` limit.

This setting also ensures that garbage collection occurs on the connection. Collection is triggered by incoming messages and heartbeats. If the size of messages can vary widely or there is not a steady stream of message traffic, can use this parameter to ensure that collection occurs.

When omitted or zero, `client_heartbeat_server` is disabled.

clock_sync_interval

`clock_sync_interval = seconds`

Periodically send the EMS server's Coordinated Universal Time (UTC) time to clients. This allows EMS clients to update their offset.

The time specified, in seconds, determines the interval at which clock sync commands are sent from the server to its clients.

When omitted or zero, the EMS server sends the offset time only when the EMS client connects to the server. If `clock_sync_interval` is -1, the offset is never sent, not even on connect. Clients do not adjust their time values to match the server time.

server_timeout_client_connection

`server_timeout_client_connection = limit`

In a server-to-client connection, if the server does not receive a heartbeat for a period exceeding this limit (in seconds), it closes the connection.

We recommend setting this value to approximately 3 times the heartbeat interval,

as it is specified in `client_heartbeat_server`.



If you do not set the `client_heartbeat_server` parameter when a `server_timeout_client_connection` is specified, a configuration error is generated during startup. If `CONFIG_ERRORS` is part of the `startup_abort_list`, the server will not start. If not, the error is printed but the server starts, and clients will be disconnected after `server_timeout_client_connection` seconds.

Zero is a special value, which disables heartbeat detection in the server (although clients still send heartbeats).

server_heartbeat_server

```
server_heartbeat_server = interval
```

In a server-to-server connection, this server sends heartbeats at this interval (in seconds). The two servers can be connected either by a route, or as a fault-tolerant pair.

server_timeout_server_connection

```
server_timeout_server_connection = limit
```

In a server-to-server connection, if this server does not receive a heartbeat for a period exceeding this limit (in seconds), it closes the connection. This parameter applies to connections from other routes and to the backup server connection.

We recommend setting this value to approximately 3.5 times the heartbeat interval of the other server. When the other server or the network are heavily loaded, or when client programs send very large messages, we recommend a larger multiple.



In a fault-tolerant configuration, the `server_timeout_server_connection` parameter has no effect on the backup server following a switchover. The backup server activates only after the timeout set by the `ft_activation` parameter.

server_heartbeat_client

```
server_heartbeat_client = interval
```

In a server-to-client connection, the server sends heartbeats to all clients at this interval (in seconds).

When omitted or zero, the default is 5 seconds.



This parameter is new in release 4.4; it is disabled when either entity is from an earlier release.

client_timeout_server_connection

```
client_timeout_server_connection = limit
```

In a server-to-client connection, if a client does not receive a heartbeat for a period exceeding this limit (in seconds), it closes the connection.

We recommend setting this value to approximately 3.5 times the heartbeat interval.

Zero is a special value, which disables heartbeat detection in the client (although the server still sends heartbeats).



This parameter is new in release 4.4; it is disabled when either entity is from an earlier release.

Fault Tolerance Parameters

See [Chapter 19, Fault Tolerance, on page 485](#) for more information about these parameters.

The fault tolerance parameters that begin with the prefix `ft_ssl` are used to secure communications between pairs of fault tolerant servers. See [SSL on page 500](#) for additional information about this process.

ft_active

```
ft_active = URL
```

Specifies the URL of the active server. If this server can connect to the active server, it will act as a backup server. If this server cannot connect to the active server, it will become the active server.

ft_heartbeat

```
ft_heartbeat = seconds
```

Specifies the interval (in seconds) the active server is to send a heartbeat signal to the backup server to indicate that it is still operating. Default is 3 seconds.

ft_activation

```
ft_activation = seconds
```

Activation interval (maximum length of time between heartbeat signals) which indicates that active server has failed. Set in seconds: default is 10. This interval should be set to at least twice the heartbeat interval.

For example:

```
ft_activation = 60
```



The `ft_activation` parameter is only used by the backup server after a fault-tolerant switchover. The active server uses the `server_timeout_server_connection` to detect a failed server.

ft_reconnect_timeout

```
ft_reconnect_timeout = seconds
```

The amount of time (in seconds) that a backup server waits for clients to reconnect (after it assumes the role of primary server in a failover situation). If a client does not reconnect within this time period, the server removes its state from the shared state files. The `ft_reconnect_timeout` time starts once the server has fully recovered the shared state, so this value does not account for the time it takes to recover the store files.

The default value of this parameter is 60.

ft_ssl_auth_only

```
ft_ssl_auth_only = enable | disable
```

When enabled, the server allows a fault tolerant server to request the use of SSL only for authentication (to protect user passwords). For an overview of this feature, see [SSL Authentication Only on page 482](#).

When disabled, the server ignores requests for this feature. When absent, the default value is disabled.

ft_ssl_identity

```
ft_ssl_identity = pathname
```

The path to a file that contains the certificate in one of the supported formats. The supported formats are PEM, DER, or PKCS#12.

See [File Names for Certificates and Keys on page 469](#) for more information on file types for digital certificates.

ft_ssl_issuer

`ft_ssl_issuer = chain_member`

Certificate chain member for the server. Supply the entire chain, including the CA root certificate. The server reads the certificates in the chain in the order they are presented in this parameter.

The certificates must be in PEM, DER, PKCS#7, or PKCS#12 format. See [File Names for Certificates and Keys on page 469](#) for more information on file types for digital certificates.

ft_ssl_private_key

`ft_ssl_private_key = key`

The server's private key. If it is included in the digital certificate in `ft_ssl_identity`, then this parameter is not needed.

This parameter supports private keys in the following formats: PEM, DER, PKCS#12.

You can specify the actual key in this parameter, or you can specify a path to a file that contains the key. See [File Names for Certificates and Keys on page 469](#) for more information on file types for digital certificates.

ft_ssl_password

`ft_ssl_password = password`

Private key or password for private keys.

You can set passwords by way of the `tibemsadmin` tool. When passwords are set with this tool, the password is obfuscated in the configuration file. See [Chapter 6, Using the EMS Administration Tool, on page 123](#) for more information about using `tibemsadmin` to set passwords.

ft_ssl_trusted

`ft_ssl_trusted = trusted_certificates`

List of trusted certificates. This sets which Certificate Authority certificates should be trusted as issuers of the client certificates.

The certificates must be in PEM, DER, or PKCS#7 format. You can either provide the actual certificates, or you can specify a path to a file containing the certificate chain.

See [File Names for Certificates and Keys on page 469](#) for more information on file types for digital certificates.

ft_ssl_rand_egd

`ft_ssl_rand_egd = pathname`

The path for the installed entropy gathering daemon (EGD), if one is installed. This daemon is used to generate random numbers for the EMS server.

ft_ssl_verify_host

`ft_ssl_verify_host = enabled | disabled`

Specifies whether the fault-tolerant server should verify the other server's certificate. The values for this parameter are `enabled` or `disabled`. By default, this parameter is `enabled`, signifying the server should verify the other server's certificate.

When this parameter is set to `disabled`, the server establishes secure communication with the other fault-tolerant server, but does not verify the server's identity.

ft_ssl_verify_hostname

`ft_ssl_verify_hostname = enabled | disabled`

Specifies whether the fault-tolerant server should verify the name in the CN field of the other server's certificate. The values for this parameter are `enabled` and `disabled`. By default, this parameter is `enabled`, signifying the fault-tolerant server should verify the name of the connected host or the name specified in the `ft_ssl_expected_hostname` parameter against the value in the server's certificate. If the names do not match, the connection is rejected.

When this parameter is set to `disabled`, the fault-tolerant server establishes secure communication with the other server, but does not verify the server's name.

ft_ssl_expected_hostname

`ft_ssl_expected_hostname = serverName`

Specifies the name the server is expected to have in the CN field of the fault-tolerant server's certificate. If this parameter is not set, the expected name is the hostname of the server.

This parameter is used when the `ft_ssl_verify_hostname` parameter is set to `enabled`.

ft_ssl_ciphers

```
ft_ssl_ciphers = cipherSuite
```

Specifies the cipher suites used by the server; each suite in the list is separated by a colon (:). This parameter can use the OpenSSL name for cipher suites or the longer, more descriptive names.

See [Specifying Cipher Suites on page 476](#) for more information about the cipher suites available in EMS and the OpenSSL names and longer names for the cipher suites.

Message Tracking Parameters**track_message_ids**

```
track_message_ids = enabled | disabled
```

Tracks messages by message ID. Default is disabled.

Enabling this parameter allows you to display messages using the `show message messageID` command in the administration tool.

track_correlation_ids

```
track_correlation_ids = enabled | disabled
```

Tracks messages by correlation ID. Disabled by default.

Enabling this parameter allows you to display messages using the `show messages correlationID` command in the administration tool.

Multicast Parameters

See [Chapter 13, Using Multicast, on page 369](#), for more information about multicast.

multicast

```
multicast = enabled | disabled
```

Enables or disables multicast in the EMS server. For example:

```
multicast = enabled
```

By default this feature is disabled.

multicast_channels

```
multicast_channels = file
```

Specifies the configuration file where multicast channels are defined.

For example:

```
multicast_channels = mychannels.conf
```

When this parameter is not included, the EMS server looks for channel definitions in the [channels.conf](#) file.

multicast_daemon_default

```
multicast_daemon_default = tcp-port
```

Specifies the TCP port on which the EMS client will attempt to connect to the multicast daemon. For example:

```
multicast_daemon_default = 9999
```

This parameter determines the TCP port that EMS clients use to connect to the multicast daemon, and is provided in the server to centrally configure all clients. It determines the behavior of the EMS client but does not affect the multicast daemon. The multicast daemon must listen for the client on the same port that the client uses to connect. If the multicast daemon is not listening on the same port that is specified by `multicast_daemon_default`, the client will be unable to connect to the daemon and an error will occur.

To change the TCP port that the multicast daemon listens on, use the `-listen` command line argument in the daemon. See [Command Line Options on page 376](#) for more information.

When this parameter is not included, the default port is 7444.

multicast_statistics_interval

```
multicast_statistics_interval = seconds
```

Specifies how often, in seconds, multicast statistics are published to the monitoring topic `$sys.monitor.multicast.stats` for each channel. Intervals of less than 5 seconds are not supported.

For example:

```
multicast_statistics_interval = 90
```

To disable multicast statistics, set the `multicast_statistics_interval` to 0 (zero).

When this parameter is not included, the default value is 0 (disabled).

TIBCO Rendezvous Parameters

See also, [Chapter 15, Working With TIBCO Rendezvous](#), on page 401.

tibrv_transports

`tibrv_transports = enabled | disabled`

Specifies whether TIBCO Rendezvous transports defined in `transports.conf` are enabled or disabled.

Unless you explicitly set this parameter to `enabled`, the default value is `disabled`—that is, all transports are disabled and will neither send messages to external systems nor receive message from them.

TIBCO SmartSockets Parameters

See also, [Chapter 16, Working With TIBCO SmartSockets](#), on page 425.

module_path

`module_path = SmartSockets-shared-library-directory`

where *SmartSockets-shared-library-directory* is the absolute path to the directory containing the SmartSockets library files. For example:

`module_path = c:\tibco\ss\bin\i86_w32`

Please see the SmartSockets documentation to locate the directory where the appropriate shared libraries are installed.



The `module_path` parameter is also used on AIX platform installations to load the IBM JVM. Specify the directory containing the `libjvm.so` and its dependent libraries.

You can specify multiple directories (for example, to load both SmartSockets and JVM libraries). Separate paths using a colon (:) on UNIX platforms, or semicolon (;) on Windows platforms.

tibss_transports

`tibss_transports = enabled | disabled`

Specifies whether TIBCO SmartSockets transports defined in `transports.conf` are enabled or disabled.

Unless you explicitly set this parameter to `enabled`, the default value is `disabled`—that is, all transports are disabled and will neither send messages to external systems nor receive message from them.

tibss_config_dir

`tibss_config_dir = pathname`

Specifies the directory for SmartSockets configuration files and message files:

- `tal_ss.cat` is a required file of messages. If it is missing, `tibemsd` outputs a warning message.
- `tibems_ss.cm` is an optional file of SmartSockets RTclient configuration options.

When this parameter is absent, `tibemsd` searches for these files in its current working directory.

For more information about these files, see *TIBCO SmartSockets User's Guide*.

Tracing and Log File Parameters

See [Chapter 17, Monitoring Server Activity, on page 445](#) for more information about these parameters.

logfile

`logfile = pathname`

Name and location of the server log file.

If the *pathname* contains spaces, it must be enclosed in double quotes.

log_trace

`log_trace = traceOptions`

Sets the trace preference on the file defined by the `logfile` parameter. If `logfile` is not set, the values have no effect.

The value of this parameter is a comma-separated list of trace options. For a list of trace options and their meanings, see [Table 72, Server Tracing Options, on page 448](#).

You may specify trace options in three forms:

- **plain** A trace option without a prefix character replaces any existing trace options.
- **+** A trace option preceded by + adds the option to the current set of trace options.
- **-** A trace option preceded by - removes the option from the current set of trace options.

The following example sets the trace log to only show messages about access control violations.

```
log_trace=ACL
```

The next example sets the trace log to show all default trace messages, in addition to SSL messages, but ADMIN messages are not shown.

```
log_trace=DEFAULT, -ADMIN, +SSL
```

logfile_max_count

```
logfile_max_count = integer
```

Specifies the maximum number of log files to be kept. Specify any number greater than 2.

When 0 or not specified, there is no limit to the number of log files kept.

logfile_max_size

```
logfile_max_size = size [KB|MB|GB]
```

Specifies the recommended maximum log file size before the log file is rotated. Set to 0 to specify no limit. Use KB, MB, or GB for units (if no units are specified, the file size is assumed to be in bytes).

The server periodically checks the size of the current log file. If it is greater than the specified size, the file is copied to a backup and then emptied. The server then begins writing to the empty log file until it reaches the specified size again.

Backup log files are named sequentially and stored in the same directory as the current log.

console_trace

```
console_trace = traceOptions
```

Sets trace options for output to stderr. The possible values are the same as for log_trace. However, console tracing is independent of log file tracing.

If logfile is defined, you can stop console output by specifying:

```
console_trace=-DEFAULT
```

Note that important error messages (and some other messages) are always output, overriding the trace settings.

This example sends a trace message to the console when a TIBCO Rendezvous advisory message arrives.

```
console_trace=RVADV
```

client_trace

```
client_trace = {enabled|disabled} [target=location]
               [user|connid|clientid=value]
```

Administrators can trace a connection or group of connections. When this property is enabled, the server instructs each client to generate trace output for opening or closing a connection, message activity, and transaction activity. This type of tracing does not require restarting the client program.

Each client sends trace output to *location*, which may be either `stderr` (the default) or `stdout`.



You can also direct client tracing output to a file, using the `tibems_SetTraceFile`, `Tibjms.setTraceFile` and `Tibems.SetTraceFile` in the C, Java and .NET libraries, respectively.

The default behavior is to trace all connections. You can specify either `user`, `connid` or `clientid` to selectively trace specific connections. The *value* can be a user name or ID (as appropriate).

Setting this parameter using the administration tool does not change its value in the configuration file `tibemsd.conf`; that is, the value does not persist across server restarts unless you set it in the configuration file.

trace_client_host

```
trace_client_host = [hostname|address|both|both_with_port]
```

Trace statements related to connections can identify the host by its hostname, its IP address, or both. When absent, the default is `hostname`. The `both_with_port` option displays the ephemeral port used on the host as well as the IP address and hostname.

Statistic Gathering Parameters

See [Chapter 17, Monitoring Server Activity, on page 445](#) for more information about these parameters.

server_rate_interval

```
server_rate_interval = seconds
```

Sets the interval (in seconds) over which overall server statistics are averaged. This parameter can be set to any positive integer greater than zero.

Overall server statistics are always gathered, so this parameter cannot be set to zero. By default, this parameter is set to 1.

Setting this parameter allows you to average message rates and message size over the specified interval.

statistics

`statistics = enabled | disabled`

Enables or disables statistic gathering for producers, consumers, destinations, and routes. By default this parameter is set to disabled.

Disabling statistic gathering resets the total statistics for each object to zero.

rate_interval

`rate_interval = seconds`

Sets the interval (in seconds) over which statistics for routes, destinations, producers, and consumers are averaged. By default, this parameter is set to 3 seconds. Setting this parameter to zero disables the average calculation.

detailed_statistics

`detailed_statistics = NONE | [PRODUCERS, CONSUMERS, ROUTES, CHANNELS]`

Specifies which objects should have detailed statistic tracking. Detailed statistic tracking is only appropriate for routes, channels, producers that specify no destination, or consumers that specify wildcard destinations. When detailed tracking is enabled, statistics for each destination are kept for the object.

Setting this parameter to NONE disabled detailed statistic tracking. You can specify any combination of PRODUCERS, CONSUMERS, ROUTES, or CHANNELS to enable tracking for each object. If you specify more than one type of detailed tracking, separate each item with a comma.

For example:

`detailed_statistics = NONE`

Turns off detailed statistic tracking.

`detailed_statistics = PRODUCERS, ROUTES`

Specifies detailed statistics should be gathered for producers and routes.

statistics_cleanup_interval

```
statistics_cleanup_interval = seconds
```

Specifies how long (in seconds) the server should keep detailed statistics if the destination has no activity. This is useful for controlling the amount of memory used by detailed statistic tracking. When the specified interval is reached, statistics for destinations with no activity are deleted.

max_stat_memory

```
max_stat_memory = size [KB|MB|GB]
```

Specifies the maximum amount of memory to use for detailed statistic gathering. If no units are specified, the amount is in bytes, otherwise you can specify the amount using KB, MB, or GB as the units.

Once the maximum memory limit is reached, the server stops collecting detailed statistics. If statistics are deleted and memory becomes available, the server resumes detailed statistic gathering.

SSL Server Parameters

See [Chapter 18, Using the SSL Protocol, on page 465](#) for more information about these parameters.

ssl_dh_size

```
ssl_dh_size = [512 | 768 | 1024 | 2048]
```

Size of the Diffie-Hellman key. Can be 512, 768, 1024, or 2048 bits. The default value is 1024.

This key is not used for cipher suites available for export.

ssl_server_ciphers

```
ssl_server_ciphers = cipherSuites
```

Specifies the cipher suites used by the server; each suite in the list is separated by a colon (:). This parameter must follow the OpenSSL cipher string syntax.

For example, you can enable two cipher suites with the following setting:

```
ssl_server_ciphers = RC4-MD5:RC4-SHA
```

See [Specifying Cipher Suites on page 476](#) for more information about the cipher suites available in EMS and the syntax for specifying them in this parameter.

ssl_require_client_cert

```
ssl_require_client_cert = enable | disable
```

If this parameter is set to `enable`, the server only accepts SSL connections from clients that have digital certificates. Connections from clients without certificates are denied.

If this parameter is set to `disable`, then connections are accepted from clients that do not have a digital certificate.

Whether this parameter is set to `enable` or `disable`, clients that do have digital certificates are always authenticated against the certificates supplied to the `ssl_server_trusted` parameter.

ssl_use_cert_username

```
ssl_use_cert_username = enable | disable
```

If this parameter is set to `enable`, a client's user name is always extracted from the CN field of the client's digital certificate, if the digital certificate is specified. If a different username is provided through the connection factory or API calls, then that username is discarded. Only the username from the CN is used.

The CN field is either a username, an email address, or a web address.



When `ssl_use_cert_username` is enabled, the username given by the CN becomes the only valid username. Any permissions associated with a different username, for example one assigned with an API call, are ignored.

ssl_cert_user_specname

```
ssl_cert_user_specname = username
```

This parameter is useful if clients are required to supply a username, but you wish to designate a special username to use when the client's username should be taken from the client's digital certificate.

For example, you may wish all clients to specify their username when logging in. This means the `ssl_use_cert_username` parameter would be set to `disable`. The username is supplied by the user, and not taken from the digital certificate. However, you may wish one username to signify that the client logging in with that name should have the name taken from the certificate. A good example of this username would be `anonymous`. All clients logging in as `anonymous` will have their user names taken from their digital certificates.

The value specified by this parameter is the username that clients will use to log in when the username should be taken from their digital certificate. A good example of the value of this parameter would be `anonymous`.

Also, the value of this parameter is ignored if `ssl_use_cert_username` is set to `enable`, in which case all client usernames are taken from their certificates. This parameter has no effect for users that have no certificate.

ssl_server_identity

`ssl_server_identity = certificate`

The server's digital certificate in PEM, DER, or PKCS#12 format. You can specify the path to a file that contains the certificate in one of the supported formats.

This parameter must be specified if any SSL ports are listed in the `listen` parameter.

PEM and PKCS#12 formats allow the digital certificate to include the private key. If these formats are used and the private key is part of the digital certificate, then setting `ssl_server_key` is optional.

For example:

`ssl_server_identity = certs/server.cert.pem`

ssl_server_key

`ssl_server_key = private_key`

The server's private key. If it is included in the digital certificate in `ssl_server_identity`, then this parameter is not needed.

This parameter supports private keys in the following formats: PEM, DER, PKCS#12.

You must specify a path to a file that contains the key.

ssl_password

`ssl_password = password`

Private key or password for private keys.

This password can optionally be specified on the command line when `tibemsd` is started.

If SSL is enabled, and the password is not specified with this parameter or on the command line, `tibemsd` will ask for the password upon startup.

You can set passwords by way of the `tibemsadmin` tool. When passwords are set with this tool, the password is obfuscated in the configuration file. See [Chapter 6, Using the EMS Administration Tool, on page 123](#) for more information about using `tibemsadmin` to set passwords.



Because connection factories do not contain the `ssl_password` (for security reasons), the EMS server uses the password that is provided in the "create connection" call for user authentication. If the create connection password is different from the `ssl_password`, the connection creation will fail.

ssl_server_issuer

```
ssl_server_issuer = chain_member
```

Certificate chain member for the server. The server reads the certificates in the chain in the order they are presented in this parameter.

The same certificate can appear in multiple places in the certificate chain.

The certificates must be in PEM, DER, PKCS#7, or PKCS#12 format.

See [File Names for Certificates and Keys on page 469](#) for more information on file types for digital certificates.

ssl_server_trusted

```
ssl_server_trusted = certificates
```

List of CA root certificates the server trusts as issuers of client certificates.

Specify only CA root certificates. Do not include intermediate CA certificates.

The certificates must be in PEM, DER, or PKCS#7 format. You can either provide the actual certificates, or you can specify a path to a file containing the certificate chain.

For example:

```
ssl_server_trusted = certs\CA1_root.pem  
ssl_server_trusted = certs\CA2_root.pem
```

See [File Names for Certificates and Keys on page 469](#) for more information on file types for digital certificates.

ssl_rand_egd

```
ssl_rand_egd = pathname
```

The path for the installed entropy gathering daemon (EGD), if one is installed. This daemon is used to generate random numbers for C clients and the EMS server. Java clients do not use this parameter.

ssl_crl_path

```
ssl_crl_path = pathname
```

A non-null value for this parameter activates the server's certificate revocation list (CRL) feature.

The server reads CRL files from this directory. The directory should contain only CRL files. If other files are located in the *pathname* directory, SSL initialization will fail.

ssl_crl_update_interval

```
ssl_crl_update_interval = hours
```

The server automatically updates its CRLs at this interval (in hours).

When this parameter is absent, the default is 24 hours.

ssl_auth_only

```
ssl_auth_only = enable | disable
```

When enabled, the server allows clients to request the use of SSL only for authentication (to protect user passwords). For an overview of this feature, see [SSL Authentication Only on page 482](#).

When disabled, the server ignores client requests for this feature. When absent, the default value is disabled.

fips140-2

```
fips140-2 = true | false
```

When true, the EMS server is enabled to run in FIPS 140-2 compliant mode.

When false or excluded, the server is not FIPS compliant. For more information, see [Enabling FIPS Compliance on page 483](#).

LDAP Parameters

See [Chapter 8, Authentication and Permissions, on page 265](#) for more information about these parameters.

ldap_url

URL of the external directory server. This can take the following forms:

```
LDAP://host:tcp_port
```

or

LDAPS: *//host:ssl_port*

For example:

LDAP: *//myLdapServer:1855*

ldap_principal

`ldap_principal` = *DN*

The distinguished name (DN) of the LDAP user that the EMS sever uses to bind to the LDAP server. This user must have privileges that allow it to bind and browse group users, but does not necessarily need to have administrative privileges.

For example:

`ldap_principal` = "cn=Manager"

ldap_credential

`ldap_credential` = *password*

The password associated with the user defined in the `ldap_principal` property. This value must be specified and cannot be an empty string.

ldap_cache_enabled

`ldap_cache_enabled` = *enable | disable*

Enables caching of LDAP data.

ldap_cache_ttl

`ldap_cache_ttl` = *seconds*

Specifies the maximum time (in seconds) that cached LDAP data is retained before it is refreshed.

ldap_conn_type

`ldap_conn_type` = [*ldaps | startTLS*]

Specifies the type of connection that the server uses to get LDAP information.

- When this parameter is absent, LDAP connections use TCP (non-secure). For backward compatibility, this is the default setting.
- *ldaps*—Use SSL on the LDAP connection (secure).
- *startTLS*—Use the startTLS extension to the LDAP version 3 protocol (secure).

ldap_tls_cacert_file

```
ldap_tls_cacert_file = pathname
```

This file contains the CA certificate that the EMS server trusts to sign the LDAP server's certificate.

You must provide `ldap_tls_cacert_file` in order to create secure connections.

Optionally, `ldap_tls_cacert_dir` can be used *in addition to*

`ldap_tls_cacert_file` in order to specify a directory with additional individual CA certificates.

ldap_tls_cacert_dir

```
ldap_tls_cacert_dir = pathname
```

When there are two or more CA certificates in the verify chain, the server scans this directory for CA certificates.

You must also provide `ldap_tls_cacert_file` in order to create secure connections.

`ldap_tls_cacert_dir` is an optional parameter that can be used *in addition to*

`ldap_tls_cacert_file` in order to specify a directory with additional individual CA certificates.

ldap_tls_cipher_suite

```
ldap_tls_cipher_suite = cipher_suite
```

Optional. You can specify the cipher suite to use for encryption on secure LDAP connections.

This parameter must follow the OpenSSL cipher string syntax; see [Specifying Cipher Suites on page 476](#). You must use short names when specifying the suite. For example, use DES-CBC-SHA rather than SSL_RSA_WITH_DES_CBC_SHA. Using long names results in an authorization error when connecting to a client.

In addition to the actual cipher names, you may specify cipher quality; for example:

- HIGH
- HIGH:MEDIUM

ldap_tls_rand_file

```
ldap_tls_rand_file = pathname
```

When the operating system does not include a random data feature, this file is the source of random data for encryption.

ldap_tls_cert_file

```
ldap_tls_cert_file = pathname
```

When the LDAP server requires client authentication, use the certificate in this file to identify the EMS server.

ldap_tls_key_file

```
ldap_tls_key_file = pathname
```

When the LDAP server requires client authentication, use the private key in this file.

When you plan to start the server remotely, we recommend that you do not password-encrypt the key file.

See [Chapter 8, Authentication and Permissions, on page 265](#) for more information about these parameters.

ldap_user_class

```
ldap_user_class = class_name
```

Name of the LDAP object class that stores users.

For example:

```
ldap_user_class = person
```

ldap_user_attribute

```
ldap_user_attribute = attribute
```

Name of the attribute on the user object class that holds the name of the user.

For example:

```
ldap_user_attribute = uid
```

ldap_user_base_dn

```
ldap_user_base_dn = DN
```

Base distinguished name (DN) of the LDAP tree that contains the users.

For example:

```
ldap_user_base_dn = "ou=People,dc=Corp"
```

ldap_user_scope

```
ldap_user_scope = onelevel | subtree
```

Specifies how deeply under the base DN to search for users. You can specify `onelevel` and `subtree` for this parameter. `onelevel` specifies to search only one level below the DN, `subtree` specifies to search all sub-trees.

For example:

```
ldap_user_scope = subtree
```

ldap_user_filter

```
ldap_user_filter = filter
```

Optional LDAP search filter for finding a given user name. Use `%s` as the placeholder for the user name in the filter. For example:

```
uid=%s
```

The full LDAP search grammar is specified in RFC 2254 and RFC 2251.

If unspecified, then a default search filter is generated based on the user object class and user name attribute.

ldap_all_users_filter

```
ldap_all_users_filter = filter
```

An optional LDAP search filter for finding all users beneath the user base DN.

If not specified, then a default search filter is generated based on the user object class and user name attribute.

See [Chapter 8, Authentication and Permissions, on page 265](#) for more information about these parameters.

ldap_group_base_dn

```
ldap_group_base_dn = DN
```

Base distinguished name (DN) of the LDAP tree that contains groups.

For example:

```
ldap_group_base_dn = "ou=Groups,dc=Corp"
```

ldap_group_scope

```
ldap_group_scope = onelevel | subtree
```

Specifies how deeply under the base DN to search for groups. You can specify `onelevel` and `subtree` for this parameter. `onelevel` specifies to search only one

level below the DN, subtree specifies to search all sub-trees.

For example:

```
ldap_group_scope = subtree
```

ldap_group_filter

```
ldap_group_filter = filter
```

Optional LDAP search filter for finding a group with a given group name. Use %s as the placeholder for the group name in the filter.

The full LDAP search grammar is specified in RFC 2254 and RFC 2251.

If unspecified, then a default search filter is generated based on the group object class and group attribute.

For example:

```
ldap_group_filter =
"(|(&(cn=%s)(objectClass=groupofUniqueNames))(&(cn=%s)
(objectClass=groupOfURLs)))"
```

ldap_all_groups_filter

```
ldap_all_groups_filter = filter
```

Optional LDAP search filter for finding all groups beneath the group base DN.

If unspecified, then a default search filter is generated based on the group object class and group attribute.

ldap_static_group_class

```
ldap_static_group_class = name
```

Name of the LDAP object class that stores static groups.

For example:

```
ldap_static_group_class = groupofuniqueNames
```

ldap_static_group_attribute

```
ldap_static_group_attribute = class
```

Name of the attribute on the static group object class that holds the name of the group.

For example:

```
ldap_static_group_attribute = cn
```

ldap_static_group_member_filter

```
ldap_static_group_member_filter = filter
```

Optional LDAP search filter for finding all static members of a group. Use %s as the placeholder for the group name in the filter.

The full LDAP search grammar is specified in RFC 2254 and RFC 2251.

If unspecified, then the following default search filter is generated based on the group object class and group attribute:

```
ldap_static_group_member_filter =  
"(&(<ldap_static_member_attribute>=<user  
DN>)(objectClass=<ldap_static_group_class>))"
```

ldap_static_member_attribute

```
ldap_static_member_attribute = attribute
```

Attribute of an LDAP static group object that specifies the distinguished names (DNs) of the members of the group.

For example:

```
ldap_static_member_attribute = uniquemember
```

ldap_dynamic_group_class

```
ldap_dynamic_group_class = class
```

Name of the LDAP object class that stores dynamic groups.

For example:

```
ldap_dynamic_group_class = groupofURLs
```

ldap_dynamic_group_attribute

```
ldap_dynamic_group_attribute = attribute
```

Name of the attribute on the dynamic group object class that holds the name of the group. For example:

```
ldap_dynamic_group_attribute = cn
```

ldap_dynamic_member_url_attribute

`ldap_dynamic_member_url_attribute = attribute`

Attribute of the dynamic LDAP group object that specifies the URLs of the members of the dynamic group.

For example:

`ldap_dynamic_member_url_attribute = memberURL`

Extensible Security Parameters

The extensible security feature allows you to write your own authentication and permissions modules for the server. For more information on this feature, see [Chapter 9, Extensible Security, on page 289](#).

jaas_classpath

`jaas_classpath = classpath`

Includes the JAR files and dependent classes used by the JAAS LoginModule. This parameter is required to enable the extensible security feature for authentication.

For example:

`jaas_classpath = ./usr/local/custom/user_jaas_plugin.jar`

jaas_config_file

`jaas_config_file = file-name`

Specifies the location of the JAAS configuration file used by the EMS server to run a custom authentication LoginModule. For more information, see [Loading the LoginModule in the EMS Server on page 294](#).

This parameter is required to enable the extensible security feature for authentication.

For example:

`jaas_config_file = jaas.conf`

jaas_login_timeout

`jaas_login_timeout = milliseconds`

Specifies the length of time, in milliseconds, that the EMS server will wait for the JAAS authentication module to execute and respond. This timeout is used each time the server passes a username and password to the LoginModule. If the module does not return a response, the server denies authentication.

This parameter is optional. If it is not included, the default timeout is 500 milliseconds.

For example:

```
jaas_login_timeout = 250
```

jaci_classpath

```
jaci_classpath = classpath
```

Includes the JAR files and dependent classes used by the JACI custom permissions module. This parameter is required to enable the extensible security feature for granting permissions.

For example:

```
jaci_classpath = ./usr/local/custom/user_jaci_plugin.jar
```

jaci_class

```
jaci_class = class-name
```

Specifies the name of the class that implements the extensible permissions interface. The class must be written using the Java Access Control Interface (JACI). For more information about writing a custom application using JACI to grant permissions, see [Writing a Permissions Module on page 300](#).

For example:

```
jaci_class = com.userco.auth.CustomAuthorizer
```

jaci_timeout

```
jaci_timeout = milliseconds
```

Specifies the length of time, in milliseconds, that the EMS server will wait for the JACI permissions module to execute and respond. This timeout is used each time the server passes a destination, username, and action to the permissions module. If the module does not return a response, the server denies authorization.

This parameter is optional. If it is not included, the default timeout is 500 milliseconds.

For example:

```
jaci_timeout = 250
```

JVM Parameters

These parameters enable and configure the Java virtual machine (JVM) in the EMS server. For more information on how JVM work in EMS, see [Enabling the JVM on page 302](#).

jre_library

`jre_library = path`

Enables the JVM in the EMS server, where *path* is the absolute path to the JRE shared library file that is installed with the JRE. Depending on your platform, this could be `jvm.dll`, `libjvm.so`, `JavaVM`, and so forth. Note that a 32-bit `tibemspd` must point to a 32-bit JVM, and a 64-bit `tibemspd` must point to a 64-bit JVM.

If this parameter is not included, the JVM is disabled by default.

If the *path* contains any spaces, the path must be enclosed in quotation marks.

For example:

```
jre_library = "C:\Program Files\Java\jdk1.6.0_04\jre\bin\server\jvm.dll"
```

jre_option

`jre_option = JVMoption`

Passes command line options to the JVM at start-up. The `jre_option` parameter can be used to define Java system properties, which are used by applications running in the JVM, such as extensible security modules.

You can use multiple `jre_option` entries in order to pass more than one options to the JVM. Permitted values for *JVMoption* include most JVM options that are defined by Sun Microsystems.

For example, this restricts the maximum heap size of the JVM to 256 megabytes:

```
jre_option = -Xmx256m
```

Using Other Configuration Files

In addition to the main configuration file, there are several other configuration files used for various purposes:

Table 33 Configuration Files

Configuration File	Description	See Page
acl.conf	Defines EMS access control lists.	239
bridges.conf	Defines bridges between destinations.	240
channels.conf	Defines the multicast channels over which multicast messages are broadcast.	241
durables.conf	Defines static durable subscribers.	244
factories.conf	Defines the connection factories stored as JNDI names on the EMS server.	245
groups.conf	Defines EMS groups.	249
jaas.conf	Locates and loads the LoginModule.	250
queues.conf	Defines EMS Queues.	250
routes.conf	Defines routes between this and other EMS servers	251
stores.conf	Defines the locations, either store files or a database, where the EMS server will store messages.	253
tibrvcm.conf	Defines the TIBCO Rendezvous certified messaging (RVCN) listeners for use by topics that export messages to a <code>tibrvcm</code> transport.	257
topics.conf	Defines EMS Topics.	257
transports.conf	Defines transports used by EMS to import messages from or export messages to external message service, such as Rendezvous and SmartSockets.	258
users.conf	Defines EMS users.	262

These configuration files can be edited by hand, but you can also use the

administration tool or the administration APIs to modify some of these files. See [Chapter 6, Using the EMS Administration Tool, on page 123](#) for more information about using the administration tool.

The following sections describe the configuration files.

acl.conf

This file defines all permissions on topics and queues for all users and groups. The format of the file is:

```
TOPIC=topic USER=user PERM=permissions
TOPIC=topic GROUP=group PERM=permissions
QUEUE=queue USER=user PERM=permissions
QUEUE=queue GROUP=group PERM=permissions
ADMIN USER=user PERM=permissions
ADMIN GROUP=group PERM=permissions
```

Table 34 ACL Parameters

Parameter Name	Description
TOPIC	Name of the topic to which you wish to add permissions.
QUEUE	Name of the queue to which you wish to add permissions.
ADMIN	Specifies that you wish to add administrator permissions.
USER	Name of the user to whom you wish to add permissions.
GROUP	Name of the group to which you wish to add permissions. The designation all specifies a predefined group that contains all users.
PERM	Permissions to add. The permissions which can be assigned to queues are send, receive and browse. The permissions which can be assigned to topics are publish, subscribe and durable and use_durable. The designation all specifies all possible permissions. For information about these permissions, refer to When Permissions Are Checked on page 286 and Inheritance of Permissions on page 81 . Administration permissions are granted to users to perform administration activities. See Administrator Permissions on page 267 for more information about administration permissions.

Example

```
ADMIN USER=sys-admins PERM=all
TOPIC=foo USER=user2 PERM=publish,subscribe
TOPIC=foo GROUP=group1 PERM=subscribe
```

bridges.conf

This file defines bridges between destinations. See [Destination Bridges on page 82](#) for more information about destination bridges.

The format of the file is:

```
[destinationType:destinationName] # mandatory -- include brackets
destinationType=destinationToBridgeTo1 [selector="msg-selector"]
destinationType=destinationToBridgeTo2 [selector="msg-selector"]
...
```

The *destination-name* can be any specific destination or a wildcard pattern to match multiple destinations.

Table 35 Bridge Parameters

Parameter Name	Description
<i>destinationType</i>	The type of the destination. That is, topic or queue.
<i>destinationName</i>	The name of the destination.
<i>destinationToBridgeTo</i>	One or more names of destinations to which to create a bridge.
<i>selector</i>	<p>This optional property specifies a message selector to limit the messages received by the bridged destination.</p> <p>For detailed information about message selector syntax, see the 'Message Selectors' section in description for the <code>Message</code> class in <i>TIBCO Enterprise Message Service Java API Reference</i>.</p>

Example

```
[topic:myTopic1]
  topic=myTopic2
  queue=myQueue1
```

channels.conf

This file defines the multicast channels over which messages published to multicast-enabled topics are broadcast. Each channel defined in this file has a unique name, and can have a different multicast address, multicast port, and property values.

The format of the file is:

```
[multicast-channel-name]
  address = multicast-group-address : multicast-port
  [ttl = hops]
  [priority = priority]
  [maxrate = size [KB|MB|GB]]
  [maxtime = seconds]
  [interface = ip-address]
```

Table 36 Channel Parameters

Parameter Name	Description
[multicast-channel-name]	[multicast-channel-name] is the name that identifies this multicast channel. Note that the square brackets [] DO NOT indicate that the <i>multicast-channel-name</i> is an option; they must be included around the name.
address	Determines where messages will be sent, where: <ul style="list-style-type: none"><i>multicast-group-address</i> is the multicast group IP address to which messages will be sent. The address must be between 224.0.0.0 and 239.255.255.255.<i>multicast-port</i> is the multicast port destination to which messages will be sent. The multicast port must be between 1 and 65535. For example, this will cause messages sent over the channel to be directed to the IP address 234.5.6.7 and multicast port 99: address = 234.5.6.7:99

Table 36 Channel Parameters

Parameter Name	Description
<code>ttl</code>	<p>Specifies the maximum number of hops that messages can make between the server and the multicast daemon.</p> <p>The number of hops between the server and multicast daemon is one plus the number of routers between them. For example, if the server and multicast daemon are in the same subnet, then there is one hop between them. If the server and multicast daemon are separated by a router, then there are two hops between them. Therefore, a <code>ttl</code> value of 1 means that the multicast data will remain on the local subnet while a <code>ttl</code> value of 2 will allow the messages to travel through one router into the next subnet.</p> <p>When this parameter is absent, the default maximum network hops allowed is 16.</p>
<code>priority</code>	<p>Specifies the channel's transmission priority when bandwidth is allocated. <code>priority</code> is given as a numerical ranking, where the highest priority is -5 and the lowest is 5.</p> <p>When this parameter is absent, the default priority is 0 (zero).</p>
<code>maxrate</code>	<p>Specifies the maximum rate at which messages can be transmitted over the channel. You can specify units of KB or MB.</p> <p>When this parameter is absent, the default value is 12 . 5MB.</p> <p>Note that a large number of retransmissions due to consumer loss can result in the publisher's allotted bandwidth being exceeded, even though the publisher is publishing below the configured <code>maxrate</code>.</p>

Table 36 Channel Parameters

Parameter Name	Description
maxtime	<p>Specifies the maximum length of time, in seconds, that the server will retain sent messages for retransmission. Messages are retransmitted when a multicast daemon detects a lost message and sends a negative acknowledgement to the EMS server.</p> <p>Note that a long maxtime will increase the amount of memory used by the server. The maximum amount of memory used by a channel will be maxrate * maxtime. For example, specifying a maxrate of 10MB and a maxtime of 10 seconds may require the server to buffer 100 megabytes of data for retransmissions.</p> <p>When this parameter is absent, messages are kept for 35 seconds.</p>
interface	<p>Specifies the IP address over which the server will send multicast traffic on this channel.</p> <p>The IP address must be a multicast capable interface. On UNIX systems, you can determine whether an IP interface is multicast capable by running the ifconfig UNIX command.</p> <p>When this parameter is not included, the default value is 0.0.0.0, which causes the EMS server to use the system's default interface.</p>

Example

```
[channel-1]
  address=234.5.6.7:99
  maxrate=10MB
  maxtime=10
  ttl=4

[channel-2]
  address=234.5.3.9:99
  maxrate=15MB
  maxtime=10
  ttl=3
```

durables.conf

This file defines static durable subscribers.

The file consists of lines with either of these formats:

```
topic-name durable-name
  [route]
  [clientid=id]
  [nolocal]
  [selector="msg-selector"]
```

Table 37 Durable Subscriber Parameters

Parameter Name	Description
topic-name	The topic of the durable subscription.
durable-name	The name of the durable subscriber.
route	When present, the subscriber is another server, and the durable-name is the name of that server. When this property is present, no other properties are permitted.
clientid=id	The client ID of the subscriber’s connection.
nolocal	When present, the subscriber does not receive messages published from its own connection.
selector="string"	When present, this selector narrows the set of messages that the durable subscriber receives. For detailed information about message selector syntax, see the ‘Message Selectors’ section in description for the Message class in <i>TIBCO Enterprise Message Service Java API Reference</i> .

Example

```
topic1 dName1
topic2 dName2 clientid=myId,nolocal
topic3 dName3 selector="urgency in ('high','medium')"
topic4 Paris route
```

Conflicting Specifications

When the server detects an conflict between durable subscribers, it maintains the earliest specification, and outputs a warning. Consider these examples:

- A static specification in this file takes precedence over a new durable dynamically created by a client.

- An existing durable dynamically created by a client takes precedence over a new static durable defined by an administrator.
- A static durable subscription takes precedence over a client attempting to dynamically unsubscribe (from the same topic and durable name).

Conflict can also arise because of wildcards. For example, if a client dynamically creates a durable subscriber for topic `foo.*`, and an administrator later attempts to define a static durable for topic `foo.1`, then the server detects this conflict and warns the administrator.

Configuration To configure durable subscriptions in this file, we recommend using the `create durable` command in the `tibemsadmin` tool; see [create durable on page 131](#).

If the `create durable` command detects an existing dynamic durable subscription with the same topic and name, it promotes it to a static subscription, and writes a specification to the file `durables.conf`.

factories.conf

This file defines the connection factories for the internal JNDI names.
The file consists of factory definitions with this format:

```
[factory-name] # mandatory -- square brackets included
  type = generic|xageneric|topic|queue|xatopic|xqueue|
  url = url-string
  metric = connections | byte_rate
  clientID = client-id
  [connect_attempt_count|connect_attempt_delay|
  connect_attempt_timeout|reconnect_attempt_count|
  reconnect_attempt_delay|reconnect_attempt_timeout = value]
  [ssl-prop = value]*
```

Table 38 Connection Factory Parameters

Parameter Name	Description
Mandatory Parameters	
These parameters are required. Values given to these parameters cannot be overridden using API calls.	
[factory-name]	[factory-name] is the name of the connection factory. Note that the square brackets [] DO NOT indicate that the <i>factory-name</i> is optional; they must be included around the name.

Table 38 Connection Factory Parameters

Parameter Name	Description
type	<p>Type of the connection factory. The value can be:</p> <ul style="list-style-type: none">• generic: Generic connection• xageneric: Generic XA connection• topic: Topic connection• queue: Queue connection• xatopic: XA topic connection• xaqueue: XA queue connection
url	<p>This string specifies the servers to which this factory creates connections:</p> <ul style="list-style-type: none">• A single URL specifies a unique server. For example: tcp://host1:8222• A pair of URLs separated by a comma specifies a pair of fault-tolerant servers. For example: tcp://host1:8222,tcp://backup1:8222• A set of URLs separated by vertical bars specifies a load balancing among those servers. For example: tcp://a:8222 tcp://b:8222 tcp://c:8222• You can combine load balancing with fault tolerance. For example: tcp://a1:8222,tcp://a2:8222 tcp://b1:8222,tcp://b2:8222 <p>This example defines two servers (a and b), each of which has a fault-tolerant backup. The client program checks the load on the primary a server and the primary b server, and connects to the one that has the smaller load. If it cannot connect to one of the primary servers, the client attempts to connect to the secondary server. For example, if it cannot connect to b1, it connects to b2.</p> <p>The connection URL cannot exceed 1000 characters.</p> <p>For cautionary information, see Load Balancing on page 249.</p>

Table 38 Connection Factory Parameters

Parameter Name	Description
Optional Parameters	
These parameters are optional. The values of these parameters can be overridden using API calls.	
<code>metric</code>	<p>The factory uses this metric to balance the load among a group of servers:</p> <ul style="list-style-type: none"> <code>connections</code>—Connect to the server with the fewest client connections. <code>byte_rate</code>—Connect to the server with the lowest byte rate. Byte rate is a statistic that includes both inbound and outbound data. <p>When this parameter is absent, the default metric is <code>connections</code>.</p> <p>For cautionary information, see Load Balancing on page 249.</p>
<code>clientID</code>	The factory associates this client ID string with the connections that it creates. The client ID cannot exceed 255 characters in length.
<code>connect_attempt_count</code>	A client program attempts to connect to its server (or in fault-tolerant configurations, it iterates through its URL list) until it establishes its first connection to an EMS server. This property determines the maximum number of iterations. When absent, the default is 2.
<code>connect_attempt_delay</code>	When attempting a first connection, the client sleeps for this interval (in milliseconds) between attempts to connect to its server (or in fault-tolerant configurations, iterations through its URL list). When absent, the default is 500 milliseconds.
<code>connect_attempt_timeout</code>	When attempting to connect to the EMS server, you can set this connection timeout period to abort the connection attempt after a specified period of time (in milliseconds).
<code>reconnect_attempt_count</code>	After losing its server connection, a client program configured with more than one server URL attempts to reconnect, iterating through its URL list until it re-establishes a connection with an EMS server. This property determines the maximum number of iterations. When absent, the default is 4.

Table 38 Connection Factory Parameters

Parameter Name	Description
reconnect_attempt_delay	When attempting to reconnect, the client sleeps for this interval (in milliseconds) between iterations through its URL list. When absent, the default is 500 milliseconds.
reconnect_attempt_timeout	When attempting to reconnect to the EMS server, you can set this connection timeout period to abort the connection attempt after a specified period of time (in milliseconds).
multicast_daemon	<p>Use the parameter to specify the TCP port that the client will use when establishing a connection to the multicast daemon.</p> <p>This parameter determines the behavior of the EMS client but does not affect the multicast daemon. The multicast daemon must listen for the client on the same port that the client uses to connect. To change the TCP port that the multicast daemon listens on, use the <code>-listen</code> command line argument in the daemon. See Command Line Options on page 376 for more information.</p> <p>See Chapter 13, Using Multicast for information on multicast.</p>
multicast_enabled	<p>Use this property to disable multicast in the connection factory.</p> <p>By default, a connection factory is always multicast-enabled if the EMS server to which it is connecting is enabled for multicast. If a client does not wish to receive messages over multicast from a multicast-enabled server, then this property can be set to disabled:</p> <ul style="list-style-type: none">• <code>enabled</code>—multicast is enabled in the factory.• <code>disabled</code>—multicast is disabled in the factory <p>See Chapter 13, Using Multicast, on page 369 for more information on multicast.</p>
ssl-prop	<p>SSL properties for connections that this factory creates.</p> <p>For further information on SSL, refer to Chapter 18, Using the SSL Protocol, page 465.</p>

Example

```
[north_america]
  type = topic
  url = tcp://localhost:7222,tcp://server2:7222
  clientID = "Sample Client ID"
  ssl_verify_host = disabled
```

Configuration To configure connection factories in this file, we recommend using the tibemsadmin tool; see [create factory on page 131](#).

Load Balancing



Do not specify load balancing in situations with durable subscribers.

If a client program that creates durable subscriber connects to server A using a load-balanced connection factory, then server A creates and supports the durable subscription. If the client program exits and restarts, and this time connects to server B, then server B creates and supports a new durable subscription—however, pending messages on server A remain there until the client reconnects to server A.



Do not specify load balancing when your application requires strict message ordering.

Load balancing chooses from among multiple servers, which inherently violates strict ordering.

groups.conf

This file defines all groups. The format of the file is:

```
group-name1 : "description"
  user-name1
  user-name2
group-name2 : "description"
  user-name1
  user-name2
```

Table 39 Group Parameters

Parameter Name	Description
<i>group-name</i>	The name of the group. The group name cannot exceed 127 characters in length.
<i>description</i>	A string describing the group.
<i>user-name</i>	One or more users that belong to the group.

Example

```
administrators: "TIBCO Enterprise Message Service administrators"
  admin
  Bob
```

jaas.conf

This file directs the TIBCO Enterprise Message Service server to the JAAS LoginModule. See [Loading the LoginModule in the EMS Server on page 294](#) for more information about the `jaas.conf` file.

queues.conf

This file defines all queues. The format of the file is:

```
[jndi-name1, jndi-name2, ...]queue-name property1, property2, ...
```



Note that, while including JNDI names is optional, the square brackets `[]` must be included around JNDI names if they are included. For more information about setting JNDI names, see [create jndiname on page 131](#).

For example, you might enter:

```
test store=mystore,secure,prefetch=2
```

Only queues listed in this file or queues with names that match the queues listed in this file can be created by the applications (unless otherwise permitted by an entry in [acl.conf](#)). For example, if queue `foo.*` is listed in this file, queues `foo.bar` and `foo.baz` can be created by the application.

Properties of the queue are inherited by all static and dynamic queues with matching names. For example, if `test.*` has the property [secure](#), then `test.1` and `test.foo` are also secure. For information on properties that can be assigned to queues, see [Destination Properties on page 58](#).

For further information on the inheritance of queue properties, refer to [Wildcards * and > on page 77](#) and [Inheritance of Properties on page 80](#).



In the sample file, a `>` wildcard at the beginning of the file allows the applications to create valid queues with any name. A `>` at the beginning of the queue configuration file means that name-matching is not required for creation of queues.

Restrictions and rules on queue names are described in [Destination Name Syntax on page 56](#).

routes.conf

This file defines routes between this TIBCO Enterprise Message Service server and other TIBCO Enterprise Message Service servers.



Routes may only be configured administratively, using the administration tool (see [Chapter 6 on page 123](#)), or the administration APIs (see `com.tibco.tibjms.admin.RouteInfo` in the online documentation). Directly editing the `routes.conf` file causes errors.

The format of the file is:

```
[route-name] # mandatory -- square brackets included.  
  url=url-string  
  zone_name=zone_name  
  zone_type=zone_type  
  [selector]*  
  [ssl-prop = value]*
```

Table 40 Route Parameters

Parameter Name	Description
[route-name]	[route-name] is the name of the passive server (at the other end of the route); it also becomes the name of the route. Note that the square brackets [] DO NOT indicate that the route-name is an option; they must be included around the name.
url	The URL of the server to and from which messages are routed.
zone_name	<p>The route belongs to the routing zone with this name. When absent, the default value is <code>default_mhop_zone</code> (this default yields backward compatibility with configurations from releases earlier than 4.0).</p> <p>You can set this parameter when creating a route, but you cannot subsequently change it.</p> <p>For further information, see these sections:</p> <ul style="list-style-type: none">• Zone on page 514• Configuring Routes and Zones on page 518

Table 40 Route Parameters

Parameter Name	Description
zone_type	<p>The zone type is either 1hop or mhop. When omitted, the default value is mhop.</p> <p>You can set this parameter when creating a route, but you cannot subsequently change it.</p> <p>The EMS server will refuse to start up if the zone type in the routes.conf file does not match the zone type already created in the \$sys.meta file that holds the shared state for the primary and backup server.</p>
selector	<p>Topic selectors (for incoming_topic and outgoing_topic parameters) control the flow of topics along the route.</p> <p>For syntax and semantics, see Selectors for Routing Topic Messages on page 525.</p>
ssl-prop	<p>SSL properties for this route.</p> <p>For further information on SSL, refer to Chapter 18, Using the SSL Protocol, page 465.</p>

Example

```
[test_route_2]
url = tcp://server2:7222
ssl_verify_host = disabled
```

stores.conf

This file defines the locations, either store files, mstore, or a database, where the EMS server will store messages or metadata (if the default `$sys.meta` definition is overridden). You can configure one or many stores in the `stores.conf` file.

Each store configured is either a file-based store, mstore, or a database store. File-based store and mstore parameters are described here. Database store parameters are described in [Chapter 10, Using Database Stores](#).

The format of the file is:

```
[store_name] # mandatory -- square brackets included
    type=file
    file=name
    file_destination_defrag=size
    [file_crc=true|false]
    [file_minimum=value]
    [file_truncate=value]
    [mode=async|sync]
    [processor_id = processor id]

[store_name]
    type=mstore
    file=name
    [processor_id = processor-id]
    [scan_iter_interval=time msec|sec|min|hour|day]
    [scan_target_interval=time msec|sec|min|hour|day]
```

Table 41 Store File Parameters

Parameter Name	Description
<code>[store_name]</code>	<p><code>[store_name]</code> is the name that identifies this store file configuration.</p> <p>Note that the square brackets <code>[]</code> DO NOT indicate that the <code>store_name</code> is an option; they must be included around the name.</p>
<code>type</code>	<p>Identifies the store type. This parameter is required for all store types. The <code>type</code> can be:</p> <ul style="list-style-type: none"> <code>file</code> — for file-based stores. <code>mstore</code> — for mstores. <code>dbstore</code> — for database stores. <p>For information about the parameters used to configure database stores, see Configuration in stores.conf on page 307.</p>

Table 41 Store File Parameters

Parameter Name	Description
file	<p>The filename that will be used when creating this store file. This parameter is required for both <code>file</code> and <code>mstore</code> types. For example, <code>mystore.db</code>.</p> <p>The location for this file can be specified using absolute or relative path names. If no path separators are present, the file will be saved in the location specified by the <code>store</code> parameter in the <code>tibemsd.conf</code> file, if any is specified there.</p>
processor_id	<p>When specified, the EMS Server binds the storage thread of this store to the specified processor.</p> <p>Do not use this parameter if the default behavior provides sufficient throughput. If no processor ID is specified for a store, the store is not bound to a specific processor.</p> <p>Specify the <i>processor-id</i> as an integer. The processor ID is numbered starting at 0 and continuing to the number of processors available, minus 1. For example, if you have four processors, the available processor IDs are 0, 1, 2, and 3.</p> <p>This parameter has similar requirements, limitations, and benefits as the <code>processor_ids</code> parameter in <code>tibemsd.conf</code>. For use guidelines, see Performance Tuning on page 121.</p>
File-Based Store Parameters	
file_destination_defrag	<p>This parameter specifies a maximum batch size used by the destination defrag feature.</p> <p>Destination defrag improves store file performance by maintaining contiguous space for new messages, while improving server read performance. When persistent pending messages begin to accumulate in a queue, messages are grouped into a batch that is re-written to disk. Messages are written close together, allowing the server to read them more efficiently when later delivering the messages to consumers.</p> <p>Specify <i>size</i> in bytes, KB, MB or GB.</p> <p>The <i>size</i> should be set to a size that is known to be acceptable for the disk where the store points to. For instance, if it is set to 2MB, your disk must be able to write a 2MB batch efficiently.</p> <p>If <code>file_destination_defrag</code> is zero or absent, the destination defrag feature is disabled.</p>

Table 41 Store File Parameters

Parameter Name	Description
<code>file_crc</code>	<p>This parameter specifies whether the EMS server uses CRC to validate data integrity when reading the store files.</p> <p>When this parameter is absent, the default is <code>true</code>.</p>
<code>file_minimum</code>	<p>This parameter preallocates disk space for the store file. Preallocation occurs when the server first creates the store file.</p> <p>You can specify units of MB or GB. Zero is a special value, which specifies no minimum preallocation. Otherwise, the value specified must be greater than 4MB.</p> <p>For example:</p> <pre>file_minimum = 32MB</pre> <p>If <code>file_truncate</code> is set to <code>true</code>, the <code>file_minimum</code> parameter prevents the EMS server from truncating the file below the set size.</p> <p>When this parameter is absent, there is no default minimum preallocation.</p>
<code>file_truncate</code>	<p>Determines whether the EMS server will occasionally attempt to truncate the store file, relinquishing unused disk space.</p> <p>When <code>file_truncate</code> is <code>true</code>, the store file may be truncated, but not below the size set in <code>file_minimum</code>.</p> <p>When this parameter is absent, the default is <code>true</code>, and the server will periodically attempt to truncate the store file.</p>
<code>mode</code>	<p>The mode determines whether messages will be written to the store file synchronously or asynchronously. Mode is either:</p> <ul style="list-style-type: none"> • <code>async</code> — the server stores messages in this file using asynchronous I/O calls. • <code>sync</code> — the server stores messages in this file using synchronous I/O calls. <p>When absent, the default is <code>async</code>.</p>

Table 41 Store File Parameters

Parameter Name	Description
mstore Parameters	
scan_iter_interval	<p>Determines the length of time between each interval of the store scan. The EMS server begins scanning a new section of the mstore at the time interval specified here.</p> <p>Specify <i>time</i> in units of <i>msec</i>, <i>sec</i>, <i>min</i>, <i>hour</i> or <i>day</i> to describe the time value as being in milliseconds, seconds, minutes, hours, or days, respectively. For example:</p> <pre>scan_iter_interval=100msec</pre> <p>By default, the mstore examines stores every 10 seconds.</p> <p>For more information, see Understanding mstore Intervals on page 33.</p>
scan_target_interval	<p>Controls the approximate length of time taken to complete a full scan of the mstore.</p> <p>Specify <i>time</i> in units of <i>msec</i>, <i>sec</i>, <i>min</i>, <i>hour</i> or <i>day</i> to describe the time value as being in milliseconds, seconds, minutes, hours, or days, respectively. For example:</p> <pre>scan_target_interval=12hour</pre> <p>By default, the scan interval is 24 hours.</p> <p>For more information, see Understanding mstore Intervals.</p>

Example

```
[my_sync]
type = file
file = /var/local/tibems/my_sync.db
file_destination_defrag=2MB
file_crc = true
file_minimum = 10MB
file_truncate = true
mode = sync
```

Example

```
[mstore1]
type = mstore
file = /var/local/tibems/mstore1.db
scan_iter_interval=100msec
scan_target_interval=12hour
```

tibrvcm.conf

This file defines the TIBCO Rendezvous certified messaging (RVCN) listeners for use by topics that export messages to a `tibrvcm` transport. The server preregisters these listeners when the server starts up so that all messages (including the first message published) sent by way of the `tibrvcm` transport are guaranteed. If the server does not preregister the RVCN listeners before exporting messages, the listeners are created when the first message is published, but the first message is not guaranteed.

The format of this file is

```
transport listenerName subjectName
```

Table 42 RVCN Listener Parameters

Parameter Name	Description
<i>transport</i>	The name of the transport for this RVCN listener.
<i>listenerName</i>	The name of the RVCN listener to which topic messages are to be exported.
<i>subjectName</i>	The RVCN subject name that messages are published to. This should be the same name as the topic names that specify the export property.

Example

```
RVCN01 listener1 foo.bar
RVCN01 listener2 foo.bar.bar
```

topics.conf

This file defines all topics. The format of the file is:

```
[jndi-name1, jndi-name2, ...]topic-name property1, property2, ...
```



Note that, while including JNDI names is optional, the square brackets [] must be included around JNDI names if they are included. For more information about setting JNDI names, see [create jndiname on page 131](#).

For example, you might enter:

```
business.inventory global, import="RV01,RV02", export="RV03",
maxbytes=1MB
```

Only topics listed in this file or topics with names that match the topics listed in this file can be created by the applications (unless otherwise permitted by an entry in [acl.conf](#)). For example, if topic `foo.*` is listed in this file, topics `foo.bar` and `foo.baz` can be created by the application.

Properties of the topic are inherited by all static and dynamic topics with matching names. For example, if `test.*` has the property [secure](#), then `test.1` and `test.foo` are also secure. For information on properties that can be assigned to topics, see [Destination Properties on page 58](#).

For further information on the inheritance of topic properties, refer to [Wildcards * and > on page 77](#) and [Inheritance of Properties on page 80](#).

Restrictions and rules on topic names are described in [Destination Name Syntax on page 56](#).

transports.conf

This file defines transports for importing messages from or exporting messages to external message services, such as TIBCO Rendezvous and TIBCO SmartSockets.

The format of the file is:

```
[transport_name] # mandatory -- square brackets included
  type = tibrv | tibrvcn | tibss # mandatory
  [topic_import_dm = TIBEMS_PERSISTENT |
                    TIBEMS_NON_PERSISTENT |
                    TIBEMS_RELIABLE]
  [queue_import_dm = TIBEMS_PERSISTENT |
                    TIBEMS_NON_PERSISTENT |
                    TIBEMS_RELIABLE]
  [export_headers = true | false]
  [export_properties = true | false]
  transport-specific-parameters
```

Table 43 Transport Parameters

Parameter Name	Description
[<i>transport_name</i>]	The name of the transport. Note that the square brackets [] DO NOT indicate that the <i>transport_name</i> is an option; they must be included around the name.

Table 43 Transport Parameters

Parameter Name	Description
<code>type</code>	<p>Transport type.</p> <ul style="list-style-type: none"> <code>tibrv</code> identifies TIBCO Rendezvous transport <code>tibrvc</code> identifies TIBCO Rendezvous Certified Messaging transport <code>tibss</code> identifies TIBCO SmartSockets transport <p>Each transport includes additional <i>transport-specific-parameters</i>:</p>
<code>topic_import_dm</code> <code>queue_import_dm</code>	<p>EMS sending clients can set the JMSDeliveryMode header field for each message. However, Rendezvous clients cannot set this header. Instead, these two parameters determine the delivery modes for all topic messages and queue messages that <code>tibemsd</code> imports on this transport.</p> <p>TIBEMS_PERSISTENT TIBEMS_NON_PERSISTENT TIBEMS_RELIABLE</p> <p>When absent, the default is TIBEMS_NON_PERSISTENT.</p>
<code>export_headers</code>	<p>When <code>true</code>, <code>tibemsd</code> includes JMS header fields in exported messages.</p> <p>When <code>false</code>, <code>tibemsd</code> suppresses JMS header fields in exported messages.</p> <p>When absent, the default value is <code>true</code>.</p>
<code>export_properties</code>	<p>When <code>true</code>, <code>tibemsd</code> includes JMS properties in exported messages.</p> <p>When <code>false</code>, <code>tibemsd</code> suppresses JMS properties in exported messages.</p> <p>When absent, the default value is <code>true</code>.</p>
<i>transport-specific-parameters</i>	See Transport-specific Parameters .



If you have multiple TIBCO Rendezvous transports configured in your `transports.conf` file, and if the EMS server fails to create a transport based on the last entry, the server will continue to traverse through the entries and attempt to create further transports.

Transport-specific Parameters

If `type = tibrv`, the extended syntax is:

```
[service = service]
[network = network]
[daemon = daemon]
[temp_destination_timeout = seconds]
[rv_queue_policy = [TIBRVQUEUE_DISCARD_NONE |
                   TIBRVQUEUE_DISCARD_FIRST |
                   TIBRVQUEUE_DISCARD_LAST] : max_msgs:qty_discard]
```

See [Rendezvous Parameters on page 405](#) for descriptions.

If `type = tibrvcn`, the extended syntax is:

```
rv_tport = name # mandatory
[cm_name = name]
[ledger_file = file-name]
[sync_ledger = true | false]
[request_old = true | false]
[explicit_config_only = true | false]
[default_ttl = seconds]
[rv_queue_policy = [TIBRVQUEUE_DISCARD_NONE |
                   TIBRVQUEUE_DISCARD_FIRST |
                   TIBRVQUEUE_DISCARD_LAST] : max_msgs:qty_discard]
```

See [Rendezvous Certified Messaging \(RVCN\) Parameters on page 406](#) for descriptions.

If `type = tibss`, the extended syntax is:

```
[username = name]
[password = password]
[server_names = single_or_list_of_servers]
[project = name]
[delivery_mode = best_effort | gmd_all | gmd_some | ordered]
[lb_mode = none | round_robin | weighted | sorted]
[override_lb_mode = enable | disable]
[gmd_file_delete = enable | disable]
[import_ss_headers = none | type_num | all]
[preserve_gmd = always | receivers | never]
```

See [SmartSockets Parameters on page 428](#) for descriptions.

Example

```
[RV01]
```

```
type = tibrv
topic_import_dm = TIBEMS_RELIABLE
queue_import_dm = TIBEMS_PERSISTENT
service = 7780
network = lan0
daemon = tcp:host5:7885
```

```
[RVCM01]
  type = tibrvc
  export_headers = true
  export_properties = true
  rv_tport = RV02
  cm_name = RVCMTrans1
  ledger_file = ledgerFile.store
  sync_ledger = true
  request_old = true
  default_ttl = 600

[SS01]
  type = tibss
  server_names = tcp:rtHost2A:5555, ssl:rtHost2B:5571
  username = emsServer6
  password = myPasswd
  project = mfg_process_control
  override_lb_mode = enable
  delivery_mode = gmd_some

[RV02]
  type = tibrv
  topic_import_dm = TIBEMS_PERSISTENT
  queue_import_dm = TIBEMS_PERSISTENT
  service = 7780
  network = lan0
  daemon = tcp:host5:7885
  rv_queue_policy = TIBRVQUEUE_DISCARD_LAST:10000:100
```

users.conf

This file defines all users. The format of the file is:

```
username:password:"description"
```

Table 44 User Parameters

Parameter Name	Description
username	The name of the user. The username cannot exceed 127 characters in length.

Table 44 User Parameters

Parameter Name	Description
<i>password</i>	<p>Leave this item blank when creating a new user. For example:</p> <pre>bob::"Bob Smith"</pre> <p>There is one predefined user, the administrator.</p> <p>User passwords are not entered in this configuration file, and remain empty (and therefore <i>not</i> secure) until you set them using the administration tool; see Assign a Password to the Administrator on page 126. You can also create users and assign passwords using API calls; see the API reference for the language you are working with.</p>
<i>description</i>	A string describing the user.

Example

```
admin::"Administrator"
Bob::"Bob Smith"
Bill::"Bill Jones"
```

After the server has started and passwords have been assigned, the file will look like this:

```
admin:$1$urmKVgq78:"Administrator"
Bob:$2$sldfkj;lsafd:"Bob Smith"
Bill:$3$tyavmwq92:"Bill Jones"
```


Chapter 8

Authentication and Permissions

You can create users and assign passwords to the users to control access to the EMS server. EMS can also be configured to use an external directory (such as an LDAP server) to control access to the server.

You can also assign permissions to users and groups to control actions that can be performed on destinations.

This chapter describes authentication and permissions in EMS.

Topics

- [EMS Access Control, page 266](#)
- [Administrator Permissions, page 267](#)
- [Enabling Access Control, page 275](#)
- [Users and Groups, page 277](#)
- [User Permissions, page 283](#)
- [When Permissions Are Checked, page 286](#)

EMS Access Control

EMS supports two basic access levels: administrative and user.

Administrator permissions control the ability of a user to login as an administrator to create, delete, or view the status of users, destinations, connections, factories, and so on. Administrators with the correct permissions can control user access to the EMS server by creating users, assigning passwords, and setting permissions.

The following procedure describes the general process for administrators to configure users, groups, and permissions and where to find more information on performing each step.

1. Enable access control for the system. See [Enabling Access Control on page 275](#).
2. Determine which destinations require access control, and enable access control for those destinations. See [Destination Control on page 276](#).
3. Determine which users need administration permissions, and decide whether administrators can perform actions globally or be restricted to a subset of actions. See [Administrator Permissions on page 267](#) for more information.
4. Determine the names of the authorized users of the system and create usernames and passwords for these users. See [Users and Groups on page 277](#).
5. Optionally, set up groups and assign users to groups. See [Users and Groups on page 277](#).
6. Optionally enable an external directory for storing users and group information. See [Configuring an External Directory on page 279](#).
7. Create the access control list by granting specific permissions to users (or groups) for destinations that need to be secure. See [User Permissions on page 283](#).

Administrator Permissions

Administrators are a special class of users that can manage the EMS server. Administrators create, modify, and delete users, destinations, routes, factories, and other items. In general, administrators must be granted permission to perform administration activities when using the administration tool or API. Administrators can be granted global permissions (for example, permission to create users or to view all queues), and administrators can be granted permissions to perform operations on specific destinations (for example, purging a queue, or viewing properties for a particular topic).



Administrator permissions control what administrators can view and change in the server only when using the administration tool or API. Administrator commands create entries in each of the configuration files (for example, `tibemsd.conf`, `acl.conf`, `routes.conf`, and so on).

You should control access to the configuration files so that only certain system administrators can view or modify the configuration files. If a user can view or modify the configuration files, setting permissions to control which destination that user can manage would not be enforced when the user manually edits the files.

Use the facilities provided by your Operating System to control access to the server's configuration files.

Administrators must be created using the administration tool, the administration APIs, or in the configuration files.

Predefined Administrative User and Group

There is a special, predefined user named `admin` that can perform any administrative action. You cannot grant or revoke any permissions to `admin`. You must assign a password for `admin` immediately after installation. For more information about changing the `admin` password, see [When You First Start tibemsaadmin on page 126](#).

There is also a special group named `$admin` for system administrator users. When a user becomes a member of this group, that user receives the same permissions as the `admin` user. You cannot grant or revoke administrator permissions from any user that is a member of the `$admin` group. You should only assign the overall system administrator(s) to the `$admin` group.

Granting and Revoking Administration Permissions

You grant and revoke administrator permissions to users using the `grant` and `revoke` commands in `tibemsadmin`, or by means of the Java or .NET admin API. You can either grant global administrator permissions or permissions on specific destinations. See [Global Administrator Permissions on page 269](#) for a complete list of global administrator permissions. See [Destination-Level Permissions on page 272](#) for a description of administrator permissions for destinations.

Global and destination-level permissions are granted and revoked separately using different administrator commands. See [Command Listing on page 128](#) for the syntax of the `grant` and `revoke` commands.

If a user has both global and destination-level administrator permissions, the actions that user can perform are determined by combining all global and destination-level administrator permissions granted to the user. For example, if an administrator is granted the `view-destination` permission, that administrator can view information about all destinations, even if the `view` permission is not granted to the administrator for specific destinations.

The `admin` user or all users in the `$admin` group can grant or revoke any administrator permission to any user. All other users must be granted the `change-admin-acl` permission and the `view-user` and/or the `view-group` permissions before they can grant or revoke administrator permissions to other users.

If a user has the `change-admin-acl` permission, that user can only grant or revoke permissions that have been granted to the user. For example, if user `BOB` is not part of the `$admin` group and he has only been granted the `change-admin-acl` and `view-user` permissions, `BOB` cannot grant any administrator permissions except the `view-user` or `change-admin-acl` permissions to other users.

Users have all administrator permissions that are granted to any group to which they belong. You can create administrator groups, grant administrator permissions to those groups, and then add users to each administrator group. The users will be able to perform any administrative action that is allowed by the permissions granted to the group to which the user belongs.

Any destination-level permission granted to a user or group for a wildcard destination is inherited for all child destinations that match the parent destination.

If protection permissions are set up, administrators can only grant or revoke permissions to other users that have the same protection permission as the administrator. See [Protection Permissions on page 273](#) for more information about protection permissions.

Enforcement of Administrator Permissions

An administrator can only perform actions for which the administrator has been granted permission. Any action that an administrator performs may be limited by the set of permissions granted to that administrator.

For example, an administrator has been granted the `view` permission on the `foo.*` destination. This administrator has not been granted the global `view-destination` permission. The administrator is only able to view destinations that match the `foo.*` parent destination. If this administrator is granted the global `view-acl` permission, the administrator is only able to view the access control list for destinations that match the `foo.*` parent. Any access control lists for other destinations are not displayed when the administrator performs the `showacl topic` or `showacl queue` commands.

If the administrative user attempts to execute a command without permission, the user may either receive an error or simply see no output. For example, if the administrator issues the `showacl queue bar.foo` command, the administrator receives a “Not authorized to execute command” error because the administrator is not authorized to view any destination except those that match `foo.*`.



An administrator can always change his/her own password, even if the administrator is not granted the `change-user` permission.

An administrator can always view his/her own permissions by issuing the:

```
showacl username
```

command, even if the administrator is not granted the `view-acl` permission.

Global Administrator Permissions

Certain permissions allow administrators to perform global actions, such as creating users or viewing all queues.

Table 45 describes the global administrator permissions.

Table 45 Global administrator permissions (Sheet 1 of 3)

Permission	Allows Administrator To...
all	Perform all administrative commands.
view-all	View any item that can be administered (for example, users, groups, topics, and so on).

Table 45 Global administrator permissions (Sheet 2 of 3)

Permission	Allows Administrator To...
change-acl	Grant and revoke user-level permissions.
change-admin-acl	Grant and revoke administrative permissions.
change-bridge	Create and delete destination bridges.
change-connection	Delete connections.
create-destination	Create any destination.
modify-destination	Modify any destination.
delete-destination	Delete any destination.
change-durable	Delete durable subscribers.
change-factory	Create, delete, and modify factories.
change-group	Create, delete, and modify groups.
change-message	Delete messages stored in the server.
change-route	Create, delete, and modify routes
change-server	Modify server parameters.
change-user	Create, delete, and modify users.
purge-destination	Purge destinations.
purge-durable	Purge durable subscribers.
shutdown	Shutdown the server.
view-acl	View user-level permissions.
view-admin-acl	View administrative permissions.
view-connection	View connections, producers and consumers.
view-bridge	View destination bridges.

Table 45 Global administrator permissions (Sheet 3 of 3)

Permission	Allows Administrator To...
<code>view-destination</code>	View destination properties and information.
<code>view-durable</code>	View durable subscribers. To view a durable subscriber, you must also have <code>view-destination</code> permission (because information about a durable subscriber includes information about the destination to which it subscribes.)
<code>view-factory</code>	View factories.
<code>view-group</code>	View all groups. Granting this permission implicitly grants <code>view-user</code> as well.
<code>view-message</code>	View messages stored in the server.
<code>view-route</code>	View routes.
<code>view-server</code>	View server configuration and information.
<code>view-user</code>	View any user.



Any type of modification to an item requires that the user can view that item. Therefore, granting any create, modify, delete, change, or purge permission implicitly grants the permission to view the associated item.

Granting the view permissions is useful when you want specific users to only be able to view items. It is not necessary to grant the view permission if a user already has a permission that allows the user to modify the item.

Global permissions are stored in the `acl.conf` file, along with all other permissions. Global permissions in this file have the following syntax:

```
ADMIN USER=<username> PERM=<permission>
```

or

```
ADMIN GROUP=<groupname> PERM=<permission>
```

For example, if a user named BOB is granted the `view-user` global administration permission and the group `sys-admins` is granted the `change-acl` permission, the following entries are added to the `acl.conf` file:

```
ADMIN USER=BOB PERM=view-user
ADMIN GROUP=sys-admins PERM=change-acl
```

Destination-Level Permissions

Administrators can be granted permissions on each destination. Destination-level permissions control the administration functions a user can perform on a specific destination. Global permissions granted to a user override any destination-level permissions.

The typical use of destination-level administration permissions is to specify permissions on wildcard destinations for different groups of users. This allows you to specify particular destinations over which a group of users has administrative control. For example, you may allow one group to control all `ACCOUNTING.* topics`, and another group to control all `PAYROLL.* queues`.

[Table 46](#) describes the destination-level administration permissions.

Table 46 Destination-level administration permissions

Permission	Allows Administrator To...
view	View information for this destination.
create	Create the specified destination. This permission is useful when used with wildcard destination names. This allows the user to create any destination that matches the specified parent.
delete	Delete this destination.
modify	Change the properties for this destination.
purge	Either purge this queue, if the destination is a queue, or purge the durable subscribers, if the destination is a topic with durable subscriptions.



Any type of modification to an item requires that the user can view that item. Therefore, granting create, modify, delete, change, or purge implicitly grants the permission to view the associated item.

Granting the view permissions is useful when you want specific users to only be able to view items. It is not necessary to grant the view permission if a user already has a permission that allows the user to modify the item.

Administration permissions for a destination are stored alongside all other permissions for the destination in the `acl.conf` file. For example, if user BOB has publish and subscribe permissions on topic `foo`, and then BOB is granted view permission, the acl listing would look like the following:

```
TOPIC=foo USER=BOB PERM=publish,subscribe,view
```



Both user and administrator permissions for a destination are stored in the same entry in the `acl.conf` file. This is for convenience rather than for clarity. User permissions specify the actions a client application can perform on a destination (publish, subscribe, send, receive, and so on). Administrator permissions specify what administrative commands the user can perform on the destination when using the administration tool or API.

Protection Permissions

Protection permissions allow you to group users into administrative domains so that administrators can only perform actions within their domain. An administrator can only perform administrative operations on a user that has the same protection permission as the user. There are four protection permissions (`protect1`, `protect2`, `protect3`, and `protect4`) that allow you to create four groups of administrators. Protection permissions do not apply to the `admin` user or users in the `$admin` group — these users can perform any action on any user regardless of protection permissions.

To use protection permissions, grant one of the protection permissions to a set of users (either individually, or to a defined group(s)). Then, grant the same protection permission to the administrator that can perform actions on those users.

For example, there are four departments in a company: sales, finance, manufacturing, and system administrators. Each of these departments has a defined group and a set of users assigned to the group. Within the system administrators, there is one manager and three other administrators, each

responsible for administering the resources of the other departments. The manager of the system administrators can perform any administrator action. Each of the other system administrators can only perform actions on members of the groups for which they are responsible.

The user name of the manager is `mgr`, the user names of the other system administrators are `admin1`, `admin2`, and `admin3`. The following commands illustrate the grants necessary for creating the example administration structure.

```
add member $admin mgr
grant admin sales protect1
grant admin admin1 protect1,all
grant admin manufacturing protect2
grant admin admin2 protect2,all
grant admin finance protect3
grant admin admin3 protect3,all
```



You can grant a protection permission, in addition to the `all` permission. This signifies that the user has all administrator privileges for anyone who also has the same protection permission. However, if you revoke the `all` permission from a user, all permissions, including any protection permissions are removed from the access control list for the user.

An administrator is able to view users that have a different protection permission set, but the administrator can only perform actions on users with the same protection permission.

For example, `admin1` can perform any action on any user in the `sales` group, and can view any users in the `manufacturing` or `finance` groups. However, `admin1` is not able to grant permissions, change passwords, delete users from, or perform any other administrative action on users of the `manufacturing` or `finance` groups. The `mgr` user is able to perform any action on any user, regardless of their protection permission because `mgr` is a member of the `$admin` group.

Enabling Access Control

Administrators can enable or disable access control for the server. Administrators can also enable and disable permission checking for specific destinations.

Server Control

The property in the main configuration file enables or disables the checking of permissions for all destinations managed by the server. The [authorization](#) property also enables or disables verification of user names and passwords.



The default setting is disabled. For secure deployments, the administrator must explicitly set authorization to enabled.

When authorization is disabled, the server grants any connection request, and does not check permissions when a client accesses a destination (for example, publishing a message to a topic).

When authorization is enabled, the server grants connections only from valid authenticated users. The server checks permissions for client operations involving secure destinations.

To enable authorization, either edit [tibemsd.conf](#) (set the `authorization` property to `enabled`, and restart the server). Or you can use the `tibemsadmin` tool to dynamically enable authorization with the following `set server` command:

```
set server authorization=enabled
```

Authorization does affect connections between fault-tolerant server pairs; see [Authorization and Fault-Tolerant Servers on page 499](#).

Administrators must always log in with the correct administration username and password to perform any administrative function—even when authorization is disabled.

Destination Control

When server [authorization](#) is enabled, the server checks user names and password of all connections without exceptions. However, operations on destinations, such as sending a message or receiving a message, are not verified unless the destination has enabled the [secure](#) property on the destination. All operations by applications on the destination with secure enabled are verified by the server according to the permissions listed in `ac1.conf`. Destinations with secure disabled continue to operate without any restrictions.



The secure property is independent of SSL-level security. The secure property controls only basic authentication and permission verification. It does not affect the security of communication between clients and server.

When a destination does not have the secure property set, any authenticated user can perform any actions on that topic or queue.

See [Destination Properties on page 58](#) for more information about destination properties.

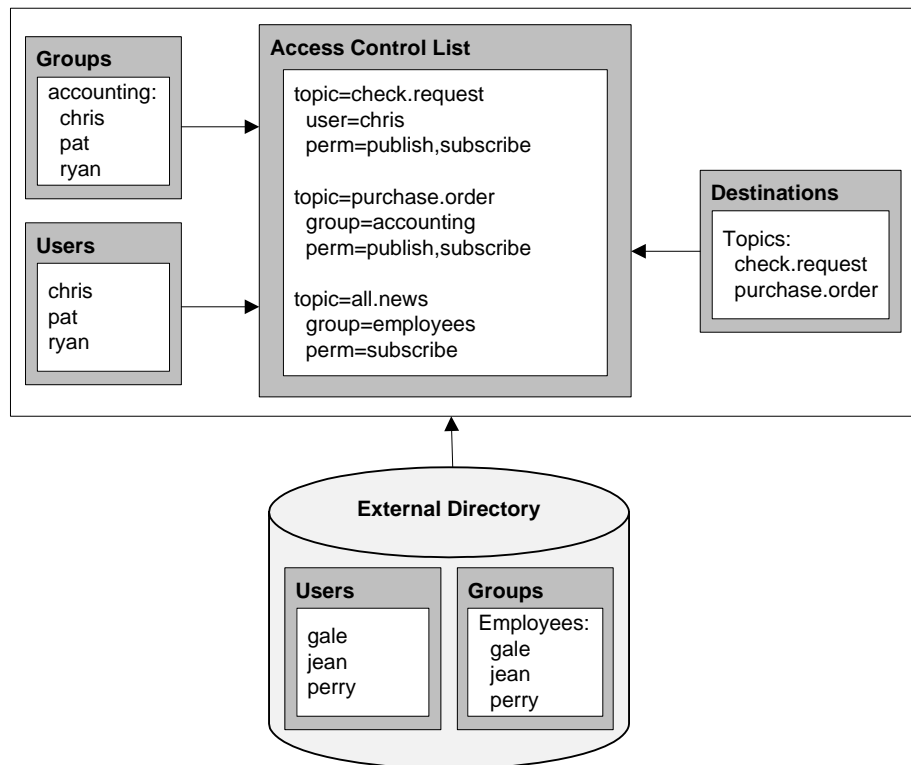
Users and Groups

User permissions apply to the activities a user can perform on each destination (topic and queue). Using permissions you can control which users have permission to send, receive, or browse messages for queues. You can also control who can publish or subscribe to topics, or who can create durable subscriptions to topics. Permissions are stored in the access control list for the server.

Groups allow you to create classes of users and control permissions on a more global level. Rather than granting and revoking permissions on destinations to individual users, you can control destination access at the group level. Users inherit any permissions from each of the groups they belong to, in addition to any permissions that are granted to them directly.

Figure 15 illustrates the relationships between users, groups and permissions.

Figure 15 Users, groups, and permissions



Externally-configured users and groups are defined and managed using the external directory. Locally-configured users and groups, as well as the access control list, are configured using any of the administration interfaces (editing configuration files, using the administration tool, or the administration APIs).



Access control and Secure Sockets Layer (SSL) have some similar characteristics. SSL allows for servers to require user authentication by way of the user's digital certificate. SSL does not, however, specify any access control at the destination level. SSL and the access control features described in this chapter can be used together or separately to ensure secure access to your system. See [Chapter 18, Using the SSL Protocol, on page 465](#) for more information about SSL.

The following sections describe users and groups in EMS.

Users

Users are specific, named IDs that allow you to identify yourself to the server. When a client logs in, the connect request should be accompanied by a username and the password associated with the username.



In special cases, you may wish to allow anonymous access to the server. In this case, a connect request does not have to supply a username or password. To configure the server to allow anonymous logins, you must create a user named `anonymous` and specify no password. Anonymous logins are not permitted unless the `anonymous` user exists.

Clients logging in anonymously are only able to perform the actions that the `anonymous` user has permission to perform.

There is one predefined user, `admin`, that performs administrative tasks, such as creating other users.

You can create and remove users and change passwords by specifying the users in the `users.conf` configuration file, using the `tibemsadmin` tool, or by using the administration APIs. For more information about specifying users in the configuration file, see [users.conf on page 262](#). For more information about specifying users using the `tibemsadmin` tool, see [Chapter 6, Using the EMS Administration Tool, on page 123](#). For more information on the administration APIs, see the online documentation.

Groups

Groups allow you to create classes of users. Groups make access control administration significantly simpler because you can grant and revoke permissions to large numbers of users with a single operation on the group. Each user can belong to as many groups as necessary. A user's permissions are the union of the permissions of the groups the user belongs to, in addition to any permissions granted to the user directly.

You can create, remove, or add users to groups by specifying the groups in `groups.conf`, using the `tibemsadmin` tool, or by using the administration APIs. For more information about specifying groups in the configuration file, see [groups.conf on page 249](#). For more information about specifying groups using the `tibemsadmin` tool, see [Chapter 6, Using the EMS Administration Tool, on page 123](#). For more information on the administration APIs, see the online documentation.

Configuring an External Directory

You can define user authentication and group information either in EMS server configuration files, or in an external directory (such as an LDAP server).

External User Authentication

EMS can be configured to authenticate users stored in an external directory server, such as an LDAP server.

The parameter `user_auth` in `tibemsd.conf` guides the EMS server when authenticating users. When a user attempts to authenticate to the EMS server, this parameter specifies the source of authentication information. This parameter can have one or more of the following values (separated by comma characters):

- `local`—obtain user authentication information from the local EMS server user configuration.
- `ldap`—obtain user authentication information from an LDAP directory server (see the LDAP-specific configuration parameters).
- `jaas`—obtain user authentication information from a custom authentication module (see [Extensible Authentication on page 292](#)).

Each time a user attempts to authenticate, the server seeks corresponding authentication information from each of the specified locations in the order that this parameter specifies. The EMS server accepts successful authentication using any of the specified sources.

Group Information

Group information stored in an external directory can also be retrieved by the EMS server. Static and dynamic groups are supported and you can configure the EMS server to retrieve either or both.

Administration Commands and External Users and Groups

You can perform administrative commands on users and groups defined either locally (in the EMS server's local configuration files) or in an external LDAP. Furthermore, you can combine users and groups that are defined in different locations (for example, you can grant and revoke permissions for users and groups defined in an LDAP, or add LDAP-defined users to locally-defined groups).



Combining authentication sources requires that the configuration parameter `user_auth` includes both `ldap` and `local`.

When you attempt to view users and groups using the `show user/s` or `show group/s` commands, any users and groups that exist in external directories have an asterisk next to their names. Users and groups from external directories will only appear in the output of these commands in the following situations:

- an externally-defined user successfully authenticates
- a user belonging to an externally-defined group successfully authenticates
- an externally-defined user has been added to a locally-defined group
- permissions on a topic or queue have been granted to an externally-defined user or group

Therefore, not all users and groups defined in the external directory may appear when the `show user/s` or `show group/s` commands are executed. Only the users and groups that meet the above criteria at the time the command is issued will appear.

You can create users and groups with the same names as externally-defined users and groups. If a user or group exists in the server's configuration and is also defined externally, the local definition of the user takes precedence.

Locally-defined users and groups will not have an asterisk by their names in the `show user/s` or `show group/s` commands.

You can also issue the `delete user` or `delete group` command to delete users and groups from the local server's configuration. The permissions assigned to the user or group are also deleted when the user or group is deleted. If you delete a user or group that is defined externally, this deletes the user or group from the server's memory and deletes any permissions assigned in the access control list,

but it has no effect on the external directory. The externally-defined user can once again log in, and the user is created in the server's memory and any groups to which the user belongs are also created. However, any permissions for the user or group have been deleted and therefore must be re-granted.

Using LDAP Directory Servers

EMS has been tested with the following external directory servers:

- Netscape/SunOne iPlanet Directory Server version 5.1
- Microsoft Active Directory shipped as part of the Windows 2000 Server

However, you should be able to use any external directory server that is compliant with LDAP V2.

The description for [tibemsd.conf on page 187](#) provides the complete list of configuration parameters for configuring an external directory server. [Table 47](#) describes parameter settings for default configurations of popular LDAP servers.

Table 47 Default configuration for popular LDAP servers (Sheet 1 of 2)

External Directory Server	Parameter Configuration
iPlanet	<pre>ldap_principal = cn=Directory Manager ldap_user_class = Person ldap_user_attribute = uid ldap_user_base_dn = ou=people, o=<your_organization> ldap_user_filter = (&(uid=%s)(objectclass=person)) ldap_group_base_dn = "ou=groups, o=<your_organization> ldap_group_filter = ((&(cn=%s)(objectclass=groupofUniqueNames))(& (cn=%s)(objectclass=groupOfURLs))) ldap_static_group_class = groupofuniquenames ldap_static_group_attribute = cn ldap_static_member_attribute = uniquemember ldap_dynamic_group_class = groupofURLs ldap_static_group_member_filter = (&(uniquemember=%s)(objectclass=groupofuniquen ames)) ldap_dynamic_group_class = groupofURLs ldap_dynamic_group_attribute = cn ldap_dynamic_member_url_attribute = memberURL</pre>

Table 47 Default configuration for popular LDAP servers (Sheet 2 of 2)

External Directory Server	Parameter Configuration
Active Directory	<pre>ldap_principal = CN=Administrator, CN=Users, DC=<your_domain> ldap_user_class = user ldap_user_attribute = cn ldap_user_filter = (&(cn=%s)(objectclass=user)) ldap_group_filter = (&(cn=%s)(objectclass=group)) ldap_static_group_class = group ldap_static_group_attribute = cn ldap_static_member_attribute = member ldap_static_group_member_filter = (&(member=%s)(objectclass=group))</pre>
Open LDAP	<pre>ldap_user_class = person ldap_user_attribute = cn ldap_user_base_dn = ou=people, dc=<your_domain_component>, dc=<your_domain_component> ldap_user_filter = (&(cn=%s)(objectclass=user)) ldap_group_base_dn = ou=groups, dc=<your_domain_component>, dc=<your_domain_component> ldap_group_filter = (&(cn=%s)(objectclass=groupofnames)) ldap_static_group_class = groupofnames ldap_static_group_attribute = cn ldap_static_member_attribute = member ldap_static_group_member_filter = (&(member=%s)(objectclass=groupofnames))</pre>
Novell	<pre>ldap_user_class = person ldap_user_attribute = cn ldap_user_base_dn = ou=people, o=<your_organization> ldap_user_filter = (&(cn=%s)(objectclass=person)) ldap_group_base_dn = ou=groups, o=<your_organization> ldap_group_filter = (&(cn=%s)(objectclass=groupofnames)) ldap_static_group_class = grouponames ldap_static_group_attribute = cn ldap_static_member_attribute = uniquemember ldap_static_group_member_filter = (&(uniquemember=%s)(objectclass=groupofnames))</pre>

User Permissions

User permissions are stored in the access control list and determine the actions a user can perform on a destination. A user's permissions are the union of the permissions granted explicitly to that user along with any permissions the user receives by belonging to a group.

When granting user permissions, you specify the user or group to whom you wish to grant the permission, the name of the destination, and the permission(s) to grant. Granting permissions is an action that is independent from both the [authorization](#) server parameter, and the [secure](#) property of the relevant destinations. The currently granted permissions are stored in the access control file, however, the server enforces them only if the [authorization](#) is enabled, and only for secure destinations.



When setting permissions for users and groups defined externally, user and group names are case-sensitive. Make sure you use the correct case for the name when setting the permissions.

User permissions can only be granted by an administrator with the appropriate permissions described in [Administrator Permissions on page 267](#).

You assign permissions either by specifying them in the [acl.conf](#) file, using the [tibemsadmin](#) tool, or by using the administration APIs. When setting user permissions, you can specify either explicit destination names or wildcard destination names. See [Inheritance of User Permissions on page 284](#) for more information on wildcard destination names and permissions.

The permissions that can be granted to users to access queues are listed in [Table 48](#); the permissions to access topics are listed in [Table 49 on page 284](#).

Table 48 Queue Permission

Name	Description
receive	permission to create queue receivers
send	permission to create queue senders
browse	permission to create queue browsers

Table 49 Topic Permission

Name	Description
subscribe	permission to create non-durable subscribers on the topic
publish	permission to publish on the topic
durable	permission to create, delete, or modify durable subscribers on the topic
use_durable	permission to use an existing durable subscriber on the topic, but <i>not</i> to create, delete, or modify the durable subscriber

Example of Setting User Permissions

The user bob has the following permission recorded in the `acl.conf` file:

```
USER=bob TOPIC=foo PERM=subscribe,publish
```

This set of permissions means that bob can subscribe to topic `foo` and publish messages to it, but bob cannot create durable subscribers to `foo`.

If bob is a member of group `engineering` and the group has the following entry in the `acl` file:

```
GROUP=engineering TOPIC=bar PERM=subscribe,publish
```

then bob can publish and subscribe to topics `foo` and `bar`.

If both the user `bob` and the group `engineering` have entries in the `acl.conf` file, then bob has permissions that are a union of all permissions set for bob directly and the permissions of the group `engineering`.

Inheritance of User Permissions

When you grant permissions to users for topics or queues with wildcard specifications, all created topics and queues that match the specification will have the same granted permissions as the permissions on the parent topic. If there are multiple parent topics, the user receives the union of all parent topic permissions for any child topic. You can add permissions to a user for topics or queues that match a wildcard specification, but you cannot remove permissions.

For example, you can grant user Bob the browse permission on queue `foo.*`. The user Bob receives the browse permission on the `foo.bar` queue, and you can also grant Bob the send permission on the `foo.bar` queue. However, you cannot take away the inherited browse permission from Bob on the `foo.bar` queue.

See [Wildcards on page 77](#) for more information about wildcards in destination names.

Revoking User Permissions

Administrators can revoke permissions for users to create consumers on a destination. Without permission, the user cannot create new consumers for a destination—however, existing consumers of the destination continue to receive messages.

You can only revoke a permission that is granted directly. That is, you cannot revoke a permission from a user that the user receives from a group. Also, you cannot revoke a permission that is inherited from a parent topic. The `revoke` command in `tibemsadmin` can only remove items from specific entries in the `acl.conf` file. The `revoke` command cannot remove items that are inherited from other entries.

You can revoke permissions in several ways:

- Remove or edit entries in the `acl.conf` file.
- Use the `revoke` commands in `tibemsadmin`; see [page 140](#).
- Use the administration APIs.

When Permissions Are Checked

If permissions are enforced (that is, the `authorization` configuration property is set, and the `secure` property is set for the destination), the server checks them when a user attempts to perform an operation on a destination. For example, create a subscription to a topic, send a message to a queue, and so on. Since permissions can be granted or revoked dynamically, the server checks them each time an operation is performed on a destination (and each time a consumer or producer is created).

For specific (non-wildcard) destination names, permissions are checked when a user performs one of the following actions:

- creates a subscription to a topic
- attempts to become a consumer for a queue
- publishes or sends a message to a topic or queue
- attempts to create queue browser

A user cannot create or send a message to a destination for which he or she has not explicitly been granted the appropriate permission. So, before creating or sending messages to the destination, a user must be granted permissions on the destination.

However, for wildcard topic names (queue consumers cannot specify wildcards), permissions are not checked when users create non-durable subscriptions. Therefore, a user can create a subscription to topic `foo.*` without having explicit permission to create subscriptions to `foo.*` or any child topics. This allows administrators to grant users the desired permissions after the user's application creates the subscriptions. You may wish to allow users to subscribe to unspecific wildcard topics, then grant permission to specific topics at a later time. Users are not able to receive messages based on their wildcard subscriptions until permissions for the wildcard topic or one or more child topics are granted.



When creating a durable subscriber, users must have the `durable` permission explicitly set for the topic they are subscribing to. For example, to create a durable subscriber to topic `foo.*`, the user must have been granted the `durable` permission to create durable subscriptions for topic `foo.*`. To subscribe an existing durable subscriber to a topic, you must have either `durable` or `use_durable` permission set on that topic.

Example of Permission Checking

This example walks through a scenario for granting and revoking permissions to a user, and describes what happens as various operations are performed.

1. User bob is working with a EMS application that subscribes to topics and displays any messages sent to those topics.
2. User bob creates a subscription to `user.*`. This topic is the parent topic of each user. Messages are periodically sent to each user (for example, messages are sent to the topic `user.bob`). Because the same application is used by many users, the application creates a subscription to the parent topic.
3. User bob creates a subscription to `topic.corp.news`. This operation fails because bob has not been granted access to that topic yet.
4. A message is sent to the topic `user.bob`, but the application does not receive the message because bob has not been granted access to the topic yet.
5. The administrator, as part of the daily maintenance for the application, grants access to topics for new users. The administrator grants the subscribe permission to topic `user.bob` and `corp.*` to user bob. These grants occur dynamically, and user bob is now able to receive messages sent to topic `user.bob` and can subscribe to topic `corp.news`.
6. The administrator sends a message on the topic `user.bob` to notify bob that access has been granted to all `corp.*` topics.
7. The application receives the new message on topic `user.bob` and displays the message.
8. User bob attempts to create a subscription for topic `corp.news` and succeeds.
9. A message is sent to topic `corp.news`. User bob's application receives this message and displays it.
10. The administrator notices that bob is a contractor and not an employee, so the administrator revokes the subscribe permission on topic `corp.*` to user bob.
The subscription to `corp.news` still exists for user bob's application, but bob cannot create any new subscriptions to children of the `corp.*` topic.

Chapter 9 **Extensible Security**

This chapter outlines how to develop and implement custom authentication and permissions modules.

Topics

- [Overview of Extensible Security, page 290](#)
- [Extensible Authentication, page 292](#)
- [Extensible Permissions, page 295](#)
- [The JVM in the EMS Server, page 302](#)

Overview of Extensible Security

The extensible security feature allows you to use your own authentication and permissions systems, in addition to the default LDAP server included in EMS, to authenticate users and authorize them to perform actions such as publish and subscribe operations. Developing custom applications to grant authentication and permissions gives you more flexibility in architecting your system.

How Extensible Security Works

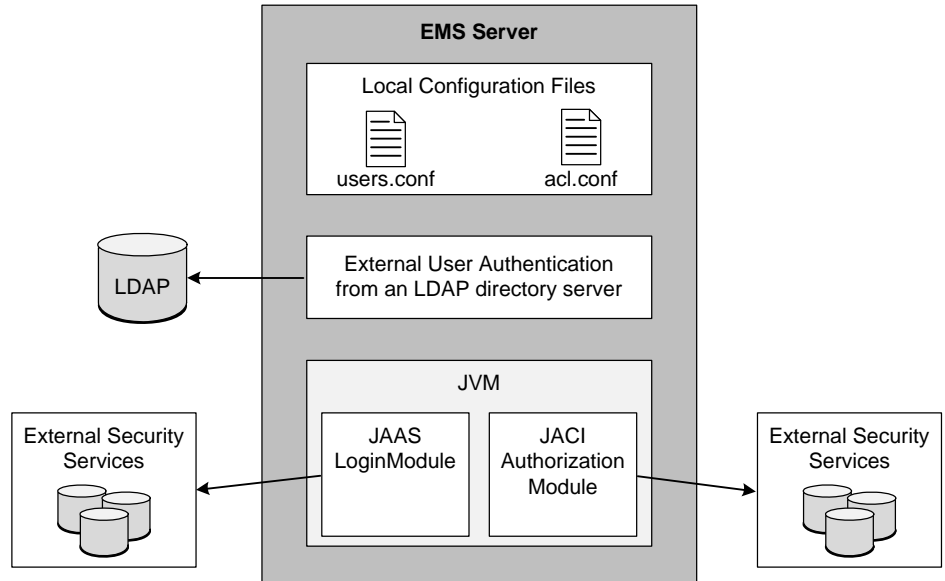
Extensible security works by allowing you to write your own authentication and permissions modules, which run in a Java virtual machine (JVM) in the EMS server. The modules connect to the server using the Java Authentication and Authorization Service (JAAS) for authentication modules, and the Java Access Control Interface (JACI) for permissions modules.

If the extensible security features are enabled when the EMS server starts, the server checks each user as it connects for authentication, and checks user permissions when they attempt to perform actions that require authorization.

Permission results are cached in the server for specified timeouts, and the permissions module is re-invoked when a cached permission expires. The server then replaces the old permission data with new data.

Extensible authentication and extensible permissions are enabled in the [tibemsd.conf](#) configuration file. Extensible security modules can connect to external security services, such as single sign on (SSO) servers or LDAP directories, which operate outside of the TIBCO Enterprise Message Service framework. Extensible security modules can work in tandem with other authorization and permissions methods, such as LDAP or the EMS [acl.conf](#) configuration file. [Figure 16 on page 291](#) shows the different security methods available in the server.

Figure 16 Methods for authenticating users and checking permissions



Extensible Authentication

The extensible authentication feature uses the Java virtual machine (JVM) and the Java Authentication and Authorization Service (JAAS) to allow you to run your own Java-based authentication module in the EMS server.

Your authentication module, or `LoginModule`, runs in the JVM within the EMS server, and is accessed by `tibemsd` using the JAAS interface. This is a flexible way to extend the security of your EMS application. The `LoginModule` can be used to augment existing authentication processes, or can be the sole method of authentication used by the EMS server. The `user_auth` parameter in the main configuration file determines when the `LoginModule` is used.

Each time an EMS client attempts to create a connection to the server, the server will authenticate the client before accepting the connection. When extensible authentication is enabled, `tibemsd` passes the username and password to the `LoginModule`, which returns an allow or deny response.

If more than one authentication mechanism is enabled, it's important to note the order that the authentication processes are employed, as determined by their order in the `user_auth` parameter. The server will search each authentication source in order, and if the user does not exist there, `tibemsd` passes the username and password to the next source.

For example, if local authentication appears before JAAS authentication, the server will search for the provided username and password first in the `users.conf` file. If the user does not exist there, `tibemsd` passes the username and password to the `LoginModule`, which allows or denies the connection attempt.

Consider a connection request from a client with the username `avogus`. If `avogus` exists in the `users.conf`, the EMS server will either authenticate or deny access to `avogus` based on the username and password located there. Only if `avogus` does not exist in the `users.conf` does the server pass the username and password to the `LoginModule`.

Enabling Extensible Authentication

Extensible authentication is enabled in the EMS server, through parameters in the `tibemsd.conf` configuration file. The required parameters are:

- `authorization`—directs the server to verify user credentials and permissions on secure destinations.
- `user_auth`—directs the EMS server to use the `LoginModule` for authentication.

- [jaas_classpath](#)—specifies the JAR files and dependent classes used by the LoginModule.
- [jaas_config_file](#)—specifies the configuration file, usually `jaas.conf`, that loads the LoginModule. For more information, see the [Example jaas.conf Configuration File on page 294](#).

Because the LoginModule runs in the Java virtual machine, you must also enable the JVM in the EMS server. See [Enabling the JVM on page 302](#) for more information.

Writing an Authentication Module

The LoginModule is a custom module that runs inside the EMS server within a JVM. The LoginModule is written using JAAS, a set of APIs provided by Sun Microsystems, and used to create pluggable Java applications. JAAS provides the interface between your code and the EMS server. JAAS is a standard part of JRE, and is installed with EMS.

LoginModule Requirements

In order to implement extensible authentication, you must write a LoginModule implementing the JAAS interface. There are some requirements for a LoginModule that will run in the EMS server:

- The LoginModule must accept the username and password from the EMS server by way of the `NameCallback` and `PasswordCallback` callbacks. The EMS server passes the username and password to the LoginModule using these callbacks, ignoring the `prompt` argument.
- If the username and password combination is invalid, the LoginModule must throw a `FailedLoginException`. The EMS server then rejects the corresponding connection attempt.
- The LoginModule must be thread-safe. That is, the LoginModule must be able to function both in a multi-threaded environment and in a single-threaded environment.
- The LoginModule should perform authentication only, by determining whether a username and password combination is valid. For information about custom permissions, see [Extensible Permissions on page 295](#).
- The LoginModule, like the Permissions Module, should not perform long operations, and should return values quickly. As these modules become part of the EMS server's message handling process, slow operations can have a severe effect on performance.
- The LoginModule must be named `EMSUserAuthentication`.

More information about JAAS, including documentation of JAAS classes and interfaces, is available through <http://java.sun.com/products/jaas/>.

Loading the
LoginModule in
the EMS Server

The EMS server locates and loads the LoginModule based on the contents of the configuration file specified by the `jaas_config_file` parameter in the `tibemsd.conf` file. Usually, the JAAS configuration file is named `jaas.conf`. This file contains the configuration information used to invoke the LoginModule.

The contents of the `jaas.conf` file should follow the JAAS configuration syntax, as documented at:

<http://java.sun.com/j2se/1.5.0/docs/api/javax/security/auth/login/Configuration.html>



The LoginModule in the JAAS configuration file must have the name `EMSUserAuthentication`.

Example `jaas.conf` Configuration File

```
EMSUserAuthentication {  
  com.tibco.tibems.tibemsd.security.example.FlatFileUserAuthLoginMod  
  ule required debug=true filename=jaas_users.txt;  
};
```


Extensible Permissions

The extensible permissions feature uses the Java virtual machine (JVM) and the Java Access Control Interface (JACI) to allow you to run your own Java-based permissions module in the EMS server.

Your Permissions Module runs in the JVM within the EMS server, and connects to `tibemspd` using the JACI interface. Like the `LoginModule`, the Permissions Module provides an extra layer of security to your EMS application. It does not supersede standard EMS procedures for granting permissions. Instead, the module augments the existing process.

When a user attempts to perform an action, such as subscribing to a topic or publishing a message, the EMS server checks the `acl.conf` file, the Permissions Module, and cached results from previous Permissions Module queries, for authorization. This process is described in detail in [How Permissions are Granted on page 296](#).

Cached Permissions

In order to speed the authorization process, the EMS server caches responses received from the Permissions Module in two pools, the *allow cache* and the *deny cache*. Before invoking the Permissions Module, the server first checks these caches for a cache entry matching the user's request.

What is Cached	<p>Each cache entry consists of a username and action, and the authorization result response from the Permissions Module:</p> <ul style="list-style-type: none"> • The username is specific; the cached permission applies only to this user. • The action is also specific. Only one action is included in each cache entry. Actions that require authorization are the same as those listed in the <code>acl.conf</code> file. • The destination can include wildcards. That is, a single cache entry can determine the user's authorization to perform the action on multiple destinations.
----------------	---

If the response from the Permissions Module authorized the action, the permission is cached in the allow cache. If the action was denied, it is cached in the deny cache.

How Long Permissions are Cached Permissions Module results also include timeouts, which determine how long the cache entry is kept in the cache before it expires. When a timeout has expired, the entry is removed from the cache. Because these timeouts are assigned by the Permissions Module, you can control how often the Permissions Module is called, and therefore how much load it puts on the EMS server.



Long timeouts on permissions cache entries can increase performance, but they also lower the system's responsiveness to changes in permissions. Consider timeout lengths carefully when writing your Permissions Module.

Administering the Cache You can view and reset cache statistics, as well as clear all cache entries. These commands are available in the administration tool:

- [jaci showstats on page 138](#)
- [jaci resetstats on page 138](#)
- [jaci clear on page 138](#)

How Permissions are Granted

When an EMS client attempts to perform an action that requires permissions, the EMS server looks in each of the following locations in turn:

1. First, the server checks the `ac1.conf` for authorization. This is the standard EMS mechanism for granting permissions, as is documented in [Chapter 8, Authentication and Permissions, on page 265](#).
2. Next, the server checks the Permissions Module allow cache for authorization. If an entry matching the username, action, and destination exists in the cache, the request is allowed.

Because destinations with wildcards can exist in the cache, an entry can have a wildcard destination that contains the requested destination. If that entry specifies the same username and action, the request is allowed. For more information on this topic, see [Implications of Wildcards on Permissions](#) below.

3. The server then checks the deny cache for a matching entry. If an entry exists in the deny cache, the request is denied.

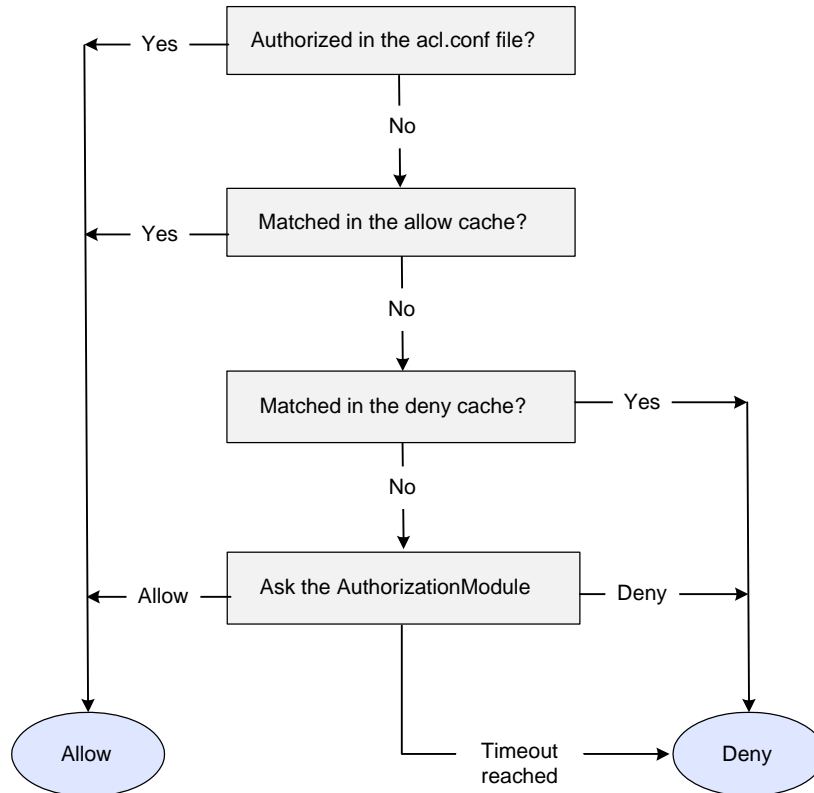
As in the allow cache, wildcards used in destinations can result in a cache entry with a destination that contains the requested destination. If that entry matches the username and action, the request is denied. For more information on this topic, see [Implications of Wildcards on Permissions](#) below.

4. Finally, if there are no matching entries in either cache, the server passes the username, action type, and destination to the Permissions Module, which returns an allow or deny authorization response. The response is also saved to the cache for the timeout specified in the response.

If the Permissions Module does not respond to the request within the timeout specified by the `jaci_timeout` parameter in the `tibemsd.conf` file, the server denies authorization by default.

Actions that require permissions are the same as those listed in the `acl.conf` file, and include operations such as subscribe to a topic and publishing to a queue. Permissions are described in [acl.conf](#) on page 239. Figure 17 shows the decision tree the server follows when granting or denying permissions.

Figure 17 The Permissions Decision Tree



In general, permissions are checked when a client initiates an operation. In the case of a browsing request, it's useful to note that the server reviews permissions only at certain points during the browsing operation.

The server checks for browsing permission when a client starts to browse a queue and whenever the client needs to refresh its list of browse-able messages. The client receives the list of messages from the server when it first begins browsing. The server refreshes the list and rechecks permissions whenever the client browses to the end of the current list.

Durable Subscribers When a durable subscriber is disconnected from the EMS server, the server continues to accumulate messages for the client. However, while the client is disconnected, there is no user associated with the durable subscriber. Because of this, the server cannot immediately check permissions for a message that is received when the client is not connected.

When a user later reconnects to the server and resubscribes to the durable subscription, the server checks permissions for the subscribe operation itself, but all messages in the backlog are delivered to the consumer without additional permission checks.

Special Circumstances There are some special circumstances under which the request, although it is not exactly matched in the `acl.conf` file, will be denied without reference to either the permissions cache or the Permissions Module. Any request will be denied if, in the `acl.conf`

- The username exists but is not associated with any destinations.
- The username exists and is associated with destinations, but not with the specific destination in the request.
- The username is part of a group, but the group is not associated with any destinations.
- The username is part of a group and the group is associated with destinations, but not with the specific destination in the request.

In general entries in the `acl.conf` file supersede entries in the Permissions Module, allowing you to optimize permission checks in well-defined static cases. When the `acl.conf` does not mention the user, the Permissions Module is fully responsible for permissions.

Implications of Wildcards on Permissions

A permission result from the Permissions Module can allow or deny the user authorization to perform the action on a range of destinations by including wildcards in the destination name. For example, even though the application attempts to have user `mw Walton` publish on topic `foo.bar.1`, the Permissions Module can grant permission to user `mw Walton` to publish messages to the topic `foo.bar.*`. For as long as this authorization is cached, `mw Walton` can also publish to the topics `foo.bar.baz` and `foo.bar.boom`, because `foo.bar.*` contains both those topics.

As long as a permission to perform an action on a destination is cached in the allow cache, the user will be authorized to perform that action, even if the permission is revoked in the external system used by the Permissions Module. This permission also extends to any destination contained by the authorized destination through the use of wildcards. The EMS server checks the allow cache

for permissions before checking the deny cache and before sending an uncached permission request to the Permissions Module. In other words, the authorization status cannot be changed until the timeout on the cache entry expires and it is removed from the cache.

Similarly, an entry in the deny cache remains there until the timeout has expired and the entry is removed. Only then does the EMS server send the request to the Permissions Module, so that a change in status can take effect.

Overlapping wildcards can make this situation even more complex. For example, consider these three destinations:

```
foo.*.baz
foo.bar.*
foo.>
```

It might seem that, if `foo.*.baz` were in a cache, then `foo.bar.*` would match it and permissions for that destination would come from the cache. In fact, however, permissions could not be determined by the cache entry, because `foo.bar.*` intersects but is not a subset of `foo.*.baz`. That is, not every destination that matches `foo.bar.*` will also match `foo.*.baz`. The destination `foo.bar.booo`, for example, would be granted permissions by `foo.bar.*`, but not by `foo.*.baz`.

Since not all destinations that `foo.bar.*` matches will also match `foo.*.baz`, we say that `foo.*.baz` intersects `foo.bar.*`. The cache entry can determine a permission if the requested destination is a subset of the cache entry, but not if it is merely an intersection. In this case, permissions cannot be determined by the cache.

The destination `foo.>`, on the other hand, contains as subsets both `foo.bar.*` and `foo.*.baz`, because any destination name that matches either `foo.bar.*` or `foo.*.baz` will also match `foo.>`. If `foo.>` is in the cache, permissions will be determined by the cache.

Enabling Extensible Permissions

Extensible permissions are enabled in the EMS server, through parameters in the `tibemspd.conf` configuration file. The required parameters are:

- `authorization`—enables authorization.
- `jaci_class`—specifies the class that implements the Permissions Module.
- `jaci_classpath`—specifies the JAR files and dependent classes used by the Permissions Module.

The Permissions Module will be used to grant permissions only to those destinations that are defined as secure in the `topics.conf` and `queues.conf` configuration files. If there are no topics or queues that include the `secure` property, then the Permissions Module will never be called because the server does not check permissions at all.

Because the Permissions Module runs in the Java virtual machine, you must also enable the JVM in the EMS server. See [Enabling the JVM on page 302](#) for more information.

Writing a Permissions Module

The Permissions Module is a custom module that runs inside the EMS server within a JVM. The Permissions Module is written using JACI, a set of APIs developed by TIBCO Software Inc. that you can use to create a Java module that will authorize EMS client requests. JACI provides the interface between your code and the EMS server. JACI is a standard component of EMS, and JACI classes and interfaces are documented in [com.tibco.tibems.tibemsd.security](#).

Requirements

In order to implement extensible permissions, you must write a Permissions Module implementing the JACI interface. There are some requirements for a Permissions Module that will run in the EMS server:

- The Permissions Module must implement the JACI Authorizer interface, which accepts information about the operation to be authorized.
- The Permissions Module must return a permission result, by way of the `AuthorizationResult` class. Permission results contain:
 - An `allowed` parameter, where `true` means that the request is allowed and `false` means the request is denied.
 - A timeout, which determines how long the permission result will be cached. Results can be cached for a time of up to 24 hours, or not at all.
 - The destination on which the user is authorized to perform the action. The destination returned can be more inclusive than the request. For example, if the user requested to subscribe to the topic `foo.bar`, the permission result can allow the user to subscribe to `foo.*`. If a destination is not included in the permission result, then the allow or deny response is limited to the originally requested destination.
 - The action type that the permission result replies to. For example, authorization to publish to the destination, or authorization to receive messages from a queue. Permissions can be granted to multiple action types, for example permission to publish and subscribe on `foo.>`. Note that the EMS server creates one cache entry for each action specified in the result.

- The Permissions Module must be thread-safe. That is, the Permissions Module must be able to function both in a multi-threaded environment and in a single-threaded environment.
- The Permissions Module, like the LoginModule, should not employ long operations, and should return values quickly. As these modules become part of the EMS server's message handling process, slow operations can have a severe effect on performance.

Documentation of JACI classes and interfaces is available through com.tibco.tibems.tibemsd.security.

The JVM in the EMS Server

The Java virtual machine (JVM) is a virtual machine on the Java platform, capable of running inside the EMS server. Select independent Java modules can operate in the JVM and plug into the EMS server. The JVM is required to use the following TIBCO Enterprise Message Service features:

- Extensible Security—see [Chapter 9, Extensible Security, on page 289](#).
- Database Stores—see [Chapter 10, Using Database Stores, page 303](#).

Enabling the JVM

The Java virtual machine is enabled in the EMS server, through parameters in the [tibemsd.conf](#) configuration file. The parameters that enable and configure the JVM are:

- [jre_library](#)—enables the JVM.
- [jre_option](#)—allows you to pass standard JVM options, defined by Sun Microsystems, to the JVM at start-up.

For more information about these parameters, see [JVM Parameters on page 237](#).



On Mac OS X platforms, you must use the 64-bit EMS server. Because Java 1.7 does not support 32-bit libraries on Mac OS X, the 32-bit EMS server cannot load the JVM.

Chapter 10 Using Database Stores

This chapter describes how to configure the TIBCO Enterprise Message Service server to store messages in a database. This chapter discusses only database stores. For more information about file-based stores, see [Store Messages in Multiple Stores on page 30](#).



The optional database store feature requires the installation and use of Hibernate Core for Java and associated jar files.

Topics

- [Database Store Overview, page 304](#)
- [Configuring Database Stores, page 305](#)
- [EMS Schema Export Tool, page 312](#)

Database Store Overview

The EMS server can connect to a database, and store messages in one or more database instances. The server connects to the database using Hibernate Core for Java to interface between the database and the EMS server.

Requirements To create database stores, you must have:

- Hibernate Core for Java and related JAR files.

Refer to the *TIBCO Enterprise Message Service Installation* manual for instructions on how to obtain and install Hibernate Core for Java.

- A database server that is supported by Hibernate, the corresponding dialect, and the appropriate JDBC driver.

The database server must be running, and the databases that the EMS server will connect to must have already been created by the database administrator.

- A username with read-write permissions and a password to the database server.

Configuring Database Stores

This section describes the steps required to configure and deploy database stores. For general conceptual information about the multiple store feature, see [Store Messages in Multiple Stores on page 30](#).

Settings for creating and configuring database stores are managed in the EMS server, and are transparent to clients. To configure the database stores feature, follow these steps:

1. Enable the database store feature in the `tibemsd.conf` by setting the parameters:

- `dbstore_classpath`
- `dbstore_driver_name`
- `dbstore_driver_dialect`
- `jre_library`

For detailed information about the `dbstore` parameters, see [Configuration in `tibemsd.conf`](#). The `jre_library` parameter, which enables the JVM in the EMS server, is described in [JVM Parameters on page 237](#).

2. Setup and configure stores in the `stores.conf` file.

You can create multiple database stores, or a combination of database and file-based stores. Each store must have a unique name. Parameters determine whether the store is a database store, provide the location of the database server, and specify the username and password that the EMS server uses to access the database.

For a list of database store parameters, see [Configuration in `stores.conf`](#) below.

3. Associate destinations with the configured stores.

Messages are sent to different stores according to their destinations. You associate a destination with a specific store using the `store` parameter in the `topics.conf` and `queues.conf` files. You can also change store associations using the `setprop topic` or `setprop queue` command in the EMS Administration Tool.

Multiple destinations can be mapped to the same store, either explicitly or using wildcards. Even if no stores are configured, the server sends persistent messages that are not associated with a store to default stores. See [Default Store Files](#) for more information.

For details about the `store` parameter, see [store on page 73](#).

4. Export database tables.

When the EMS server is configured to store messages in a database, the database schema must be exported before the server is started. Use the EMS Schema Export Tool to create, drop, and update the database tables.

For details, see [EMS Schema Export Tool on page 312](#).

Configuration in `tibemsd.conf`

These parameters are set in the `tibemsd.conf` configuration file.

`dbstore_classpath`

`dbstore_classpath = pathname`

Includes all the JAR files required by the EMS server when employing the database store feature. This parameter must be set when a store of type `dbstore` has been created in the `stores.conf` file.

Required JAR files are determined by the installed Hibernate release, and are documented in the `_README.txt` file that is located in the `lib/` directory of the Hibernate distribution. Many of these JAR files are version-specific, and the required versions may change with new Hibernate releases. You should verify the required version and modify the `dbstore_classpath` variable accordingly.

If you are using Hibernate release 3.2.5, for example, the `dbstore_classpath` should include paths to the following JAR files:

- `hibernate3.jar`
- `dom4j-1.6.1.jar`
- `commons-collections-2.1.1.jar`
- `commons-logging-1.0.4.jar`
- `ehcache-1.2.3.jar`
- `jta.jar`
- `cglib-2.1.3.jar`
- `antlr-2.7.6.jar`
- `c3p0-0.9.1.jar`
- `asm.jar`
- `asm-attrs.jar`
- Database-specific driver JAR file. Supported jar files are listed in Database Servers and Drivers in *TIBCO Enterprise Message Service Installation*.

For an example, see *EMS_HOME/samples/config/tibemsd-db.conf*.

dbstore_driver_name

`dbstore_driver_name = name`

Specifies the name of the JDBC driver used by Hibernate.

For example:

- If you are using the MySQL InnoDB database server:
`dbstore_driver_name=com.mysql.jdbc.Driver`
- If you are using the Microsoft SQL Server:
`dbstore_driver_name=com.microsoft.sqlserver.jdbc.SQLServerDriver`
- If you are using Oracle 10g:
`dbstore_driver_name=oracle.jdbc.driver.OracleDriver`
- If you are using IBM DB2 Server:
`dbstore_driver_name=com.ibm.db2.jcc.DB2Driver`

dbstore_driver_dialect

`dbstore_driver_dialect = dialect`

Specifies the Hibernate SQL dialect used to construct SQL commands.

For example, if you are using the MySQL with InnoDB database server:

`dbstore_driver_dialect = org.hibernate.dialect.MySQL5InnoDBDialect`

The SQL dialect is defined by Hibernate. For a list of databases and the associated dialects, see the *readme.txt* file located in the Hibernate install directory archive.

Configuration in stores.conf

This section describes parameters configured for each database store in the *stores.conf* file. The *stores.conf* includes definitions for both database and file-based stores. For information about configuring file-based stores, see [stores.conf on page 253](#).

The format of the file is:

```
[store_name] # mandatory -- square brackets included.
  type = dbstore
  dbstore_driver_url = JDBCURL
  dbstore_driver_username = username
  dbstore_driver_password = password
  [processor_id = processor-id]
```

Table 50 Database Store File Parameters

Parameter Name	Description
[store_name]	<p>[store_name] is the name that identifies this store configuration.</p> <p>Note that the square brackets [] DO NOT indicate that the store_name is an option; they must be included around the name.</p>
type=dbstore	<p>Identifies the store type. This parameter is required for all store types. The type can be:</p> <ul style="list-style-type: none">file — for file-based stores.mstore — for mstores.dbstore — for database stores. <p>For information about the parameters used to configure file-based stores, see stores.conf on page 253.</p>
dbstore_driver_url	<p>Provides the location of the database server. The URL entered uses the syntax specified by the JDBC driver for your database.</p> <p>Please see documentation specific to your JDBC driver for more information. If you are using an Oracle RAC database, also see Using a TAF Configured URL on page 311.</p>
dbstore_driver_username	<p>The username that the EMS server uses to access the database.</p> <p>Note that this user must have read and write permissions to the database.</p>

Table 50 Database Store File Parameters

Parameter Name	Description
dbstore_driver_password	<p>The password that the server uses, in conjunction with the username provided in <code>dbstore_driver_username</code>, to access the database.</p> <p>You can mangle this and other passwords by way of the <code>tibemsadmin</code> tool. See Table 15, tibemsadmin Options, on page 124 for more information about using <code>tibemsadmin</code> to mangle passwords.</p>
processor_id	<p>When specified, the EMS Server binds the storage thread of this store to the specified processor.</p> <p>Do not use this parameter if the default behavior provides sufficient throughput. If no processor ID is specified for a store, the store is not bound to a specific processor.</p> <p>Specify the <i>processor-id</i> as an integer. The processor ID is numbered starting at 0 and continuing to the number of processors available, minus 1. For example, if you have four processors, the available processor IDs are 0, 1, 2, and 3.</p> <p>This parameter has similar requirements, limitations, and benefits as the processor_ids parameter in <code>tibemspd.conf</code>. For use guidelines, see Performance Tuning on page 121.</p>

Example Using MySQL Server

```
[ $sys.fail-safe ]
  type=dbstore
  dbstore_driver_url=jdbc:mysql://mysqlsrv_1:3306/sysfs
  dbstore_driver_username=admin
  dbstore_driver_password=admin123

[ $sys.meta ]
  type=dbstore
  dbstore_driver_url=jdbc:mysql://mysqlsrv_1:3306/sysmeta
  dbstore_driver_username=admin
  dbstore_driver_password=admin123
```

Example Using Microsoft SQL Server

```
[ $sys.meta]
type=dbstore
dbstore_driver_url=jdbc:sqlserver://sqlsrv_1:3415;databaseName=sysmeta
dbstore_driver_username=admin
dbstore_driver_password=admin123

[ $sys.fail-safe]
type=dbstore
dbstore_driver_url=jdbc:sqlserver://sqlsrv_1:3415;databaseName=sysfs
dbstore_driver_username=admin
dbstore_driver_password=admin123
```

Example Using Oracle 10g

```
[ $sys.meta]
type=dbstore
dbstore_driver_url=jdbc:oracle:thin:adminmeta/admin123@osrv_1:1521:orclperf
dbstore_driver_username=adminmeta
dbstore_driver_password=admin123

[ $sys.fail-safe]
type=dbstore
dbstore_driver_url=jdbc:oracle:thin:adminfs/admin123@osrv_1:1521:orclperf
dbstore_driver_username=adminfs
dbstore_driver_password=admin123
```

Example Using Oracle RAC 10g

```
[ $sys.fail-safe]
type=dbstore
dbstore_driver_url=jdbc:oracle:oci:<user>/<passwd>@(DESCRIPTION=
(AADDRESS=(PROTOCOL=TCP)(HOST=<host1>)(PORT=1521))(ADDRESS=(PROTOCO
L=TCP)(HOST=<host2>)(PORT=1521))(CONNECT_DATA=(SERVICE_NAME=orcl)(
FAILOVER_MODE=(TYPE=SELECT)(METHOD=BASIC)(RETRIES=180)(DELAY=5))))
dbstore_driver_username=admin
dbstore_driver_password=admin123
```

For more information, see [Configuration for the Oracle RAC Database](#) below.

Example Using IBM DB2 Server

```
[ $sys.meta]
type=dbstore
dbstore_driver_url=jdbc:db2://db2srv_1:50000/SYSMETA
dbstore_driver_username=admin
dbstore_driver_password=admin123

[ $sys.fail-safe]
type=dbstore
dbstore_driver_url=jdbc:db2://db2srv_1:50000/SYSFS
dbstore_driver_username=admin
dbstore_driver_password=admin123
```


Configuration for the Oracle RAC Database

The TIBCO Enterprise Message Service server must connect to the Oracle RAC 10g or 11g database using the Oracle JDBC OCI driver and TAF configuration.

Installing the OCI Driver

We recommend using the Oracle Instant Client, which is an optimized light-weight OCI driver package available from Oracle:

<http://www.oracle.com/technology/tech/oci/instantclient/index.html>

Follow the instructions provided to install the Oracle Instant Client.

Using a TAF Configured URL

To ensure that the EMS server does not lose its connection to the database during a database failover, the server should connect to the database using a Transparent Application Failover (TAF) configured URL. For example:

```
jdbc:oracle:oci:@(DESCRIPTION=(ADDRESS=(PROTOCOL=TCP)(HOST=host1)(PORT=1521))(ADDRESS=(PROTOCOL=TCP)(HOST=host2)(PORT=1521))(CONNECT_DATA=(SERVICE_NAME=orcl)(FAILOVER_MODE=(TYPE=SELECT)(METHOD=BASIC)(RETRIES=180)(DELAY=5))))
```



True Transparent Application Failover is not supported. If a database failover occurs while the EMS server is performing a transactional activity, the EMS server does not replay or restart the failed transaction. However, a TAF connection allows the EMS server to recover fully as long as no transaction was taking place at the time of the failover.

EMS Schema Export Tool

Each database store that is configured for an EMS server includes a configuration parameter pointing to a database. The EMS Schema Export Tool creates and exports database tables for the database stores. Database administrators can use the Schema Export Tool to selectively export and tune schemas to suit your database and messaging system.



The EMS Schema Export Tool must be used to export database tables when one or more database stores are configured. That is, if any stores of type `dbstore` are configured, you must export the database schema before starting the EMS server.

The Schema Export Tool is a JAR file, `tibemsd_util.jar`, located in the same directory as `tibemsd`. Command line options, described in [Table 51 on page 313](#), determine whether database tables are created or dropped, and whether they are printed to the console, saved to a file, or exported to the database.

Before invoking the Schema Export Tool, you must:

- Configure the global database store parameters for the EMS server. The parameters that configure the global database store settings begin with `dbstore_`. See [Configuration in `tibemsd.conf` on page 306](#) for details about these parameters.
- Configure at least one store of type `dbstore`. See [Configuration in `stores.conf` on page 307](#) for more information about configuring database stores.

How the Schema Export Tool Works

When it is invoked, the Schema Export Tool accepts the `tibemsd.conf` or `tibemsd.json` file and reviews the database store parameters, then parses the stores configured, either in the `stores.conf` file or in the JSON configuration file. Depending on the options specified when it was invoked, the Schema Export Tool will create, drop, or update the database tables for the stores of type `dbstore` that are configured.

The tool can perform the selected actions on all database stores, or only on specific stores. The Schema Export Tool can also print the database tables it creates to the console, or export them either to the database or to a specified file.

Running the
Schema Export
Tool

The Schema Export Tool is invoked from the command line. The tool can be invoked from its directory, or by giving the absolute path to the `tibems_util.jar` file. For example:

On Windows

```
> java -jar EMS_HOME\bin\tibemsd_util.jar options
```

Or

```
> java -jar c:\tibco\ems\8.0\bin\tibemsd_util.jar options
```

On Unix

```
> java -jar EMS_HOME/bin/tibemsd_util.jar options
```

Or

```
$ java -jar /opt/tibco/ems/8.0/bin/tibemsd_util.jar options
```

This table shows the options that are used with the Schema Export Tool:

Table 51 EMS Schema Export Tool options

Option	Description
<code>-tibemsdconf</code> <i>pathname</i>	<p>The absolute path to the <code>tibemsd.conf</code> or <code>tibemsd.json</code> file. For example, on a UNIX system:</p> <pre>/opt/tibco/ems/8.0/samples/config/tibemsd.conf</pre> <p>This tool supports JSON configuration files only when run on those platforms for which Central Administration is supported. For a list of supported platforms, see the supported platforms list for Central Administration in the <i>TIBCO Enterprise Message Service Installation</i> guide.</p> <p>Text-based <code>tibemsd.conf</code> files are supported on all platforms.</p>
<code>-exporttofile</code>	<p>Export the schema to a file named <i>store-name.ddl.log</i>, where <i>store-name</i> is the name of the database store. If multiple database stores are configured, then one file is created for each database store.</p> <p>If neither <code>exporttofile</code> nor <code>export</code> option is included, the schema export tool prints the schema to the console.</p> <p>If both <code>-exporttofile</code> and <code>-export</code> are included, the Schema Export Tool exports the database schema to both locations.</p>

Table 51 EMS Schema Export Tool options

Option	Description
-export	<p>Export the schema to the database configured for the store.</p> <p>If neither <code>export</code> nor <code>exporttofile</code> option is included, the schema export tool prints the schema to the console.</p> <p>If both <code>-export</code> and <code>-exporttofile</code> are included, the Schema Export Tool exports the database schema to both locations.</p>
-store <i>storename</i> =create update drop	<p>Create, update, or drop the schema for one or more specific stores that are named in the stores configuration file.</p> <p>If you choose the <code>create</code> option for a schema that already exists, the Schema Export Tool recreates the schema.</p> <p>Note that <code>create</code> prints the schema to screen but does not deploy it. You must use <code>export</code> or <code>exporttofile</code> in order to implement the schema.</p>
-createall	<p>Create all the stores found in the stores configuration file. Note that this option drops any existing configurations when creating the new stores.</p>
-dropall	<p>Drop all the stores found in the stores configuration file.</p>
-updateall	<p>Update the schema for all stores configured in the found in stores configuration file.</p>
-help	<p>Print information about the schema export tool and its options, and exit the tool.</p>

Examples

The following examples show how the Schema Export Tool can be used to create database schemas in various configurations.

Example 1 This example shows how the Schema Export Tool can be invoked from any directory by giving the absolute path to the `tibemsd_util.jar`:

```
$ java -jar /opt/tibco/ems/8.0/bin/tibemsd_util.jar -help
```

Example 2 In this example, the Schema Export Tool creates and exports database schemas for all the stores found in the `stores.conf` that is set in the specified `tibemsd-mssqlserver.conf` file:

```
java -jar /opt/tibco/ems/8.0/bin/tibemsd_util.jar -tibemsdconf
/opt/tibco/ems/8.0/samples/config/tibemsd.conf -createall -export
```

Example 3 In this example, the Schema Export Tool exports the database schema for the `$sys.failSAFE` store to the database:

```
jar -jar /opt/tibco/ems/8.0/bin/tibemsd_util.jar -tibemsdconf
/opt/tibco/ems/8.0/samples/config/tibemsd.conf -export -store
\$sys.failSAFE=create
```

Example 3 In this example, the Schema Export Tool writes the database schema for the `$sys.failSAFE` store to the file `$sys.failSAFE.ddl.log`:

```
$ jaav -jar /opt/tibco/ems/8.0/bin/tibemsd_util.jar -tibemsdconf
/opt/tibco/ems/8.0/samples/config/tibemsd.conf -exporttofile
-store \$sys.failSAFE=create
```

Example 4 In this example the Schema Export Tool creates and exports the database schema for the store `mystore1`, but drops the schema associated with `mystore2` and exports the change:

```
java -jar /opt/tibco/ems/8.0/bin/tibemsd_util.jar -tibemsdconf
/opt/tibco/ems/8.0/samples/config/tibemsd.conf -store
mystore1=create -store mystore2=drop -export
```


Chapter 11 **Developing an EMS Client Application**

This chapter outlines how to develop EMS client applications in Java, C and C#.

Topics

- [JMS Specification, page 318](#)
- [Sample Clients, page 320](#)
- [Programmer Checklists, page 321](#)
- [Connection Factories, page 332](#)
- [Connecting to the EMS Server, page 335](#)
- [Creating a Session, page 337](#)
- [Setting an Exception Listener, page 338](#)
- [Dynamically Creating Topics and Queues, page 340](#)
- [Creating a Message Producer, page 342](#)
- [Creating a Message Consumer, page 346](#)
- [Working with Messages, page 352](#)

JMS Specification

EMS implements the JMS 2.0 specification, which is backward compatible with earlier versions of the specification.

While the old JMS 1.0.2b interfaces are still supported, newly developed applications should use the JMS 2.0 or 1.1 interfaces instead. It is recommended to avoid using 1.0.2b interfaces, in particular due to their lack of flexibility. With these, an application initially written to work with topics has to be reworked if it needs to use queues, whereas an application based on the 1.1 or 2.0 APIs relies on a generic destination infrastructure that would not need to be altered significantly.

To get a better understanding and illustration of how the various JMS objects relate to each other, refer to the [JMS Specification](#) and to the samples client applications provided with EMS.

The code examples in this chapter illustrate the use of the JMS 2.0 interface.

JMS 2.0 Specification



Note that software release 8.0.0 implements the JMS 2.0 specification in Java only.

The JMS 2.0 specification introduces several new features, including delivery delay, shared subscriptions, asynchronous sending and the Simplified API.

The Simplified API is offered in addition to the API originally provided with JMS 1.1, which is now called the Classic API. The Simplified API is less verbose than the Classic API, and introduces several important new objects:

- **JMSContext** is used to create messages, as well as JMS consumers and JMS producers. Each JMS context uses one session and one connection, but does not expose those. Additionally, multiple JMS context objects can share the same connection.
- **JMSConsumer** is a message consumer that has the ability to receive a message body without the need to use a Message object.
- **JMSProducer** is similar to an anonymous message producer, and provides a convenient API for configuring delivery options, message properties, and message headers.

Methods in the Simplified API throw unchecked exceptions rather than checked exceptions. For a sample showing the Simplified API in use, see the new Java sample file called `tibjmsJMSContextSendRecv.java`. This sample file demonstrates the Simplified API in the simplest possible way; for greater detail, refer to the Java API Reference Pages.

JMS 1.1 Specification

In the JMS 1.1 specification, applications using the point to point (queues) or publish and subscribe (topics) models use the same interfaces to create objects. The JMS specification refers to these interfaces as *common facilities* because these interfaces create objects that can be used for either topics or queues.

JMS 1.0.2b Specification

The JMS 1.0.2b specification defined specific interfaces for topics and for queues.

The JMS 1.0.2b interfaces have the same structure as the JMS 1.1 common facilities, but the interfaces are specific to topics or queues.

Sample Clients

TIBCO Enterprise Message Service includes several sample client applications that illustrate various features of EMS. You may wish to view these sample clients when reading about the corresponding features in this manual.

The samples are included in the *EMS_HOME/samples/java*, *EMS_HOME/samples/c*, and *EMS_HOME/samples/cs* subdirectories of the EMS installation directory. Each subdirectory includes a README file that describes how to compile and run the sample clients.

[Chapter 4, Getting Started, on page 93](#) walks through the procedures for setting up your EMS environment and running some of the sample clients.

Programmer Checklists

This section provides a checklist that outlines the steps for creating an EMS application in each language:

- [Java Programmer's Checklist on page 321](#)
- [C Programmer's Checklist on page 322](#)
- [C# Programmer's Checklist on page 328](#)

Java Programmer's Checklist

Install

- Install the EMS software release, which automatically includes the EMS jar files in the `EMS_HOME/lib` subdirectory.
- Add the full pathnames for the following jar files to your CLASSPATH:
`jms.jar`
`tibjms.jar`
- If SSL is used for communication, add the following file to the CLASSPATH:
`tibcrypt.jar`
- Programs that use the unshared state failover API must add the following file to the CLASSPATH:
`tibjmsufo.jar`



All jar files listed in this section are located in the `lib` subdirectory of the TIBCO Enterprise Message Service installation directory.

To use Entrust with an EMS client, you must separately purchase and install the Entrust libraries. If you use the Entrust libraries, you must include them in the CLASSPATH *before* the JSSE JAR files. To use Entrust with JDK, you must download the unlimited strength policy JAR files from Sun's website and install them in your local installation of JDK. For installation and configuration details, see Entrust documentation.

See [Security Considerations on page 116](#) for a complete discussion of what is needed for a secure deployment.

Code

Import the following packages into your EMS application:

```
import javax.jms.*;
import javax.naming.*;
```

Compile

Compile your EMS application with the `javac` compiler to generate a `.class` file.

For example:

```
javac MyApp.java
```

generates a `MyApp.class` file.

Run

Use the `java` command to execute your EMS `.class` file.

For example:

```
java MyApp
```

C Programmer's Checklist

Developers of EMS C programs can use this checklist during the five phases of the development cycle.

Install

- Install the EMS software release, which automatically includes the EMS client libraries, binaries, and header files in the `EMS_HOME/lib` subdirectory.

Code

Application programs must:

- Add `EMS_HOME/include` to the include path. (OpenVMS environments do not require an include path; skip this item.)
- Include the `tibems.h` header file:


```
#include <tibems/tibems.h>
```
- Programs that use the C administration API must also include the `emsadmin.h` header file:


```
#include <tibems/emsadmin.h>
```

- Programs that use the unshared state failover API must also include the `tibufo.h` header file:

```
#include <tibems/tibufo.h>
```
- Call `tibems_Open()` to initialize the EMS C API and `tibems_Close()` to deallocate the memory used by EMS when complete.

Compile and Link

- Compile programs with an ANSI-compliant C compiler.
- Link with the appropriate EMS C library files; see [Link These Library Files on page 323](#).

See the `samples/c/readme` file for details.

Run

- **UNIX** If you use dynamic EMS libraries on a UNIX platform, the environment variable `$LD_LIBRARY_PATH` must include the `EMS_HOME/lib` directory (which contains the shared library files). (On some UNIX platforms, this variable is called `$SHLIB_PATH` or `$SYLIB_LIBRARY_PATH`).
- **Windows** The `PATH` must include the `ems\8.0\bin` directory.
- **OpenVMS** The installation procedure automatically installs the shareable images required for using EMS dynamic libraries.
- **All Platforms** The application must be able to connect to a EMS server process (`tibemsd`).

Link These Library Files

EMS C programs must link the appropriate library files. The following sections describe which files to link for your operating system platform:

- [32-Bit UNIX on page 324](#)
- [64-Bit UNIX on page 324](#)
- [Microsoft Windows on page 325](#)
- [OpenVMS on page 327](#)

32-Bit UNIX

In 32-bit UNIX environments, both shared and static libraries are available. We recommend shared libraries to ease forward migration.

Table 52 Linker Flags for 32-Bit UNIX

Linker Flag	Description
-ltibems	All programs must link using this library flag.
-lssl -lcrypto	Programs that use SSL must link using these library flags.
-lz	Programs that use compression must link using this library flag.
-ltibemslookup -lldap -lxml2 -llber	Programs that use EMS LDAP lookup must link using these library flags.
-ltibemsadmin	Programs that use the C administration library must link using this library flag.
-ltibemsufo	Programs that use the unshared state failover library must link using this library flag.

64-Bit UNIX

In 64-bit UNIX environments, both shared and static libraries are available. We recommend shared libraries to ease forward migration. In this release, 64-bit libraries are available on HP-UX, Solaris, AIX and Linux (2.4 glibc 2.3) platforms.

To use 64-bit libraries, you must include *TIBCO_HOME/ems/8.0/lib/64* in your library path, and it must precede any other EMS directory in the library path.

Table 53 Linker Flags for 64-Bit UNIX (Sheet 1 of 2)

Linker Flag	Description
-ltibems64	All programs must link using this library flag.
-lssl -lcrypto	Programs that use SSL must link using these library flags.
-lz	Programs that use compression must link using this library flag.

Table 53 Linker Flags for 64-Bit UNIX (Sheet 2 of 2)

Linker Flag	Description
-ltibemslookup64 -lldap -lxml2 -llber	Programs that use EMS LDAP lookup must link using these library flags.
-ltibemsadmin64	Programs that use the C administration library must link using this library flag.
-ltibemsufo64	Programs that use the unshared state failover library must link using this library flag.

Microsoft Windows

For a list of Windows platforms that Release 8.0 supports, see the file `readme.txt` in the installation directory. Both DLLs and static libraries are available. We recommend DLLs to ease forward migration.

Table 54 Dynamic Library Files for Microsoft Windows

Library File	Description
With dynamic libraries (DLLs), use the <code>/MT</code> compiler option.	
tibems.lib ws2_32.lib	All programs must link these libraries.
tibemslookup.lib libxml2.lib	Programs that use EMS LDAP lookup must link these libraries.
liboldap32.lib olber32.lib libldap32_d.lib liblber32_d.lib	In addition, programs that use EMS lookup must link one of these pairs of libraries.
tibemsadmin.lib	Programs that use the C administration library must link using this library.
tibemsufo.lib	Programs that use the C unshared state failover library must link using this library.

Table 55 Static Library Files for Microsoft Windows

Library File	Description
With static libraries (DLLs), use the /MD compiler option.	
libtibems.lib ws2_32.lib ssleay32mt.lib libeay32mt.lib zlib.lib	All programs must link these libraries.
libtibemslookup.lib libxml2.lib	Programs that use EMS LDAP lookup must link this library.
liboldap32.lib olber32.lib libldap32_d.lib liblber32_d.lib	In addition, programs that use EMS lookup must link one of these pairs of libraries.
libtibemsadmin.lib	Programs that use the C administration library must link using this library.
libtibemsufo.lib	Programs that use the C unshared state failover library must link using this library.

OpenVMS

In OpenVMS environments, both shared and static libraries are available. We recommend shared libraries to ease forward migration.



When upgrading from EMS 4.3 to 4.4 or later versions, EMS client executables that were linked with the EMS 4.3 dynamic libraries (shareable images) must be relinked to the new libraries after EMS 4.4 has been installed with its associated third party libraries. The third party libraries are part of the full installation of EMS.

Table 56 Shareable Image Library Files for OpenVMS

Library File	Description
LIBTIBEMSSHR.EXE	All programs must link this library.
LIBCRYPTOSHR.EXE LIBSSLHR.EXE	Programs that use SSL must link these libraries.
LIBZSHR.EXE	Programs that use data compression must link this library.
LIBTIBEMSADMINSHR.EXE	Programs that use the C administration library must link this library.

Table 57 Static Library Files for OpenVMS

Library File	Description
LIBTIBEMS.OLB	All programs must link this library.
LIBCRYPTO.OLB LIBSSL.OLB	Programs that use SSL must link these libraries.
LIBZ.OLB	Programs that use data compression must link this library.
LIBTIBEMSADMIN.OLB	Programs that use the C administration library must link this library.

C# Programmer’s Checklist

Developers of EMS C# programs can use this checklist during the four phases of the development cycle.

Install

- Install the EMS software release, which automatically includes the EMS assembly DLLs in the *EMS_HOME\bin* subdirectory.

Code

- Import the correct EMS assembly (see [Table 58](#)).

Table 58 EMS Assembly DLL

Version	DLL
.NET API	TIBCO.EMS.dll
.NET Administration API	TIBCO.EMS.ADMIN.dll
.NET Unshared State API	TIBCO.EMS.UFO.dll

Compile

- Compile with any .NET compiler.

Run

- The EMS assembly must be in the global assembly cache (this location is preferred), or in the system path, or in the same directory as your program executable.
- To automatically upgrade to the latest .NET assemblies, include the appropriate policy file in the global cache. See [Automatic Upgrades Between Versions](#) for more information.
- The application must be able to connect to a EMS daemon process (*tibemsd*).

Assembly Versioning

TIBCO Enterprise Message Service assembly DLLs are versioned using the format `1.0.release.version`, where *release* is the EMS release number and *version* is an arbitrary value. For example, the assembly version number for software release 8.0.0 is similar to `1.0.800.8`.



Applications that use a release of EMS earlier than 6.0 do not use standard .NET versioning. Prior to TIBCO Enterprise Message Service release 6.0.0, all EMS .NET assemblies showed an assembly version number `1.0.0.0`, which allowed client applications to upgrade to the latest version of EMS without rebuilding.

This functionality is now available through the *policy* DLL files.

Automatic Upgrades Between Versions

In order to allow for seamless upgrades between releases, the TIBCO Enterprise Message Service installation includes policy and configuration files that redirect existing applications from an older assembly to the newest assembly. There is a policy and configuration file for each EMS library:

- A *policy.1.0.assembly* file. For example, `policy.1.0.TIBCO.EMS.dll`. The policy file must be included in the global cache to enable automatic upgrades.
- An *assembly.config* file. For example, `TIBCO.EMS.dll.config`. The configuration file must be present when the related policy file is added to the global cache.

Table 59 shows the policy and configuration files for each EMS assembly.

Table 59 EMS Policy Files

Version	Files
.NET API	<code>policy.1.0.TIBCO.EMS.dll</code> <code>TIBCO.EMS.dll.config</code>
.NET Administration API	<code>policy.1.0.TIBCO.EMS.ADMIN.dll</code> <code>TIBCO.EMS.ADMIN.dll.config</code>
.NET Unshared State API	<code>policy.1.0.TIBCO.EMS.UFO.dll</code> <code>TIBCO.EMS.UFO.dll.config</code>

Enabling Updates

To enable automatic updates for a library, add the appropriate policy file to the global cache. Note that the related configuration file must be located in the directory with the policy file in order to add the policy file to the global cache.

Disabling Automatic Upgrades

If you do not want your older applications to automatically move to the newer version, do not include the `policy` DLL in the global cache. When the `policy.1.0.assembly` file is absent, the client application is not upgraded.

Running Multiple Clients from Different EMS Releases

To deploy two or more applications that are built with different TIBCO Enterprise Message Service releases:

- 1. Build clients using the different .NET client assemblies.
- 2. Include all desired versions of the .NET client assemblies in the global cache.
- 3. Do *not* include the `policy` DLL in the global cache.

Excluded Features and Restrictions

This section summarizes features that are not available in the .NET library.

Note that compression, SSL, and the LDAP lookup of administered objects features are available only with Microsoft .NET Framework 2.0.

Table 60 .NET Feature Support

Feature	.NET
XA protocols for external transaction managers	—
ConnectionConsumer, ServerSession, ServerSessionPool	—
Compression	Yes
SSL	Yes
Modify socket buffer sizes (see Tibems.SetSocketReceiveBufferSize and Tibems.SetSocketSendBufferSize in the HTML reference)	Yes
Daemon threads (see Tibems.SetSessionDispatcherDaemon in the HTML reference)	Yes

Character Encoding

.NET programs represent strings within messages as byte arrays. Before sending an outbound message, EMS programs translate strings to their byte representation using an encoding, which the program specifies. Conversely, when EMS programs receive inbound messages, they reconstruct strings from byte arrays using the same encoding.

When a program specifies an encoding, it applies to all strings in message bodies (names and values), and properties (names and values). It does not apply to header names nor values. The method `BytesMessage.WriteUTF` always uses UTF-8 as its encoding.

Outbound Messages

Programs can determine the encoding of strings in outbound messages in three ways:

- Use the default global encoding, UTF-8.
- Set a non-default global encoding (for all outbound messages) using `Tibems.SetEncoding`.
- Set the encoding for an individual message using `Tibems.SetMessageEncoding`.

Inbound Messages

An inbound message from another EMS client explicitly announces its encoding. A receiving client decodes the message using the proper encoding.

For more information about character encoding, see [Character Encoding in Messages on page 36](#).

Connection Factories

A client must connect to a running instance of the EMS server to perform any JMS operations. A connection factory is an object that encapsulates the data used to define a client connection to an EMS server. The minimum factory parameters are the type of connection and the URL for the client connection to the EMS server.

A connection factory is either dynamically created by the application or obtained from a data store by means of a naming service, such as a Java Naming and Directory Interface (JNDI) server or a Lightweight Directory Access Protocol (LDAP) server.

Looking up Connection Factories

EMS provides a JNDI implementation that can be used to store connection factories. Java, C, and C# clients can use the EMS JNDI implementation to lookup connection factories.

You can also store connection factories in any JNDI-compliant naming service or in an LDAP server. Java clients can lookup connection factories in any JNDI-compliant naming service. C and C# clients use LDAP servers.

[Looking up Administered Objects Stored in EMS on page 362](#) describes how to lookup a connection factory from an EMS server. How to create connection factories in a EMS server is described in [Creating and Modifying Administered Objects in EMS on page 360](#).

Dynamically Creating Connection Factories

Normally client applications use JNDI to look up a Connection Factory object. However, some situations require clients to connect to the server directly. To connect to the EMS server directly, the application must dynamically create a connection factory.

The following examples show how to create a connection factory in each supported language for JMS connections. Each API also supports connection factories for JMS XA connections.

In each example, the `serverUrl` parameter in these expressions is a string defining the protocol and the address of the running instance of the EMS Server. The `serverUrl` parameter has the form:

```
serverUrl = protocol://host:port
```

The supported *protocols* are **tcp** and **ssl**. For example:

```
serverUrl = tcp://server0:7222
```

For a fault-tolerant connection, you can specify two or more URLs. For example:

```
serverUrl = tcp://server0:7222,tcp://server1:7344
```

See [Configuring Clients for Fault-Tolerant Connections on page 504](#) for more information. For details on using SSL for creating secure connections to the server, see [Configuring SSL in EMS Clients on page 472](#) and [Creating Connection Factories for Secure Connections on page 360](#).

Java

To dynamically create a `TibjmsConnectionFactory` object in a Java client:

```
ConnectionFactory factory = new
    com.tibco.tibjms.TibjmsConnectionFactory(serverUrl);
```

See the `tibjmsMsgProducer.java` sample client for a working example.

C

To dynamically create a `tibemsConnectionFactory` type in a C client:

```
factory = tibemsConnectionFactory_Create();
status = tibemsConnectionFactory_SetServerURL(
    factory, serverUrl);
```

See the `tibemsMsgProducer.c` sample client for a working example.

C#

To dynamically create a `ConnectionFactory` object in a C# client:

```
ConnectionFactory factory = new
    TIBCO.EMS.ConnectionFactory(serverUrl);
```

See the `csMsgProducer.cs` sample client for a working example.

Setting Connection Attempts, Timeout and Delay Parameters

By default, a client will attempt to connect to the server two times with a 500 ms delay between each attempt. A client can modify this behavior by setting new connection attempt count and delay values. There are also a number of factors that may cause a client to hang while attempting to create a connection to the EMS server, so you can set a connection timeout value to abort a connection attempt after a specified period of time. For best results, timeouts should be at least 500 milliseconds. EMS also allows you to establish separate count, delay and timeout settings for reconnections after a fault-tolerant switchover, as described in [Setting Reconnection Failure Parameters on page 505](#).

The following examples establish a connection count of 10, a delay of 1000 ms and a timeout of 1000 ms.

Java

Use the `TibjmsConnectionFactory` object's `setConnAttemptCount()`, `setConnAttemptDelay()`, and `setConnAttemptTimeout()` methods to establish new connection failure parameters:

```
factory.setConnAttemptCount(10);
factory.setConnAttemptDelay(1000);
factory.setConnAttemptTimeout(1000);
```

C

Use the `tibemsConnectionFactory_SetConnectAttemptCount` and `tibemsConnectionFactory_SetConnectAttemptDelay` functions to establish new connection failure parameters:

```
status = tibemsConnectionFactory_SetConnectAttemptCount(
    factory, 10);

status = tibemsConnectionFactory_SetConnectAttemptDelay(
    factory, 1000);

status = tibemsConnectionFactory_SetConnectAttemptTimeout(
    factory, 1000);
```

C#

Use the `ConnectionFactory.SetConnAttemptCount`, `ConnectionFactory.SetConnAttemptDelay`, and `ConnectionFactory.SetConnAttemptTimeout` methods to establish new connection failure parameters:

```
factory.setConnAttemptCount(10);
factory.setConnAttemptDelay(1000);
factory.setConnAttemptTimeout(1000);
```


Connecting to the EMS Server

A connection with the EMS server is defined by the Connection object obtained from a Connection Factory, as described in [Connection Factories on page 332](#).

A connection is a fairly heavyweight object, so most clients will create a connection once and keep it open until the client exits. Your application can create multiple connections, if necessary.

The following examples show how to create a Connection object.

Java

Use the `TibjmsConnectionFactory` object's `createConnection()` method to create a Connection object:

```
Connection connection =
    factory.createConnection(userName,password);
```

See the `tibjmsMsgProducer.java` sample client for a working example.

C

Use the `tibemsConnectionFactory_CreateConnection` function to create a connection of type `tibemsConnection`:

```
tibemsConnection connection = NULL;
status = tibemsConnectionFactory_CreateConnection(factory,
    &connection, userName, password);
```

If there is no connection factory, a C client can use the `tibemsConnection_Create` function to dynamically create a `tibemsConnection` type:

```
status = tibemsConnection_Create(&connection,
    serverUrl,NULL,userName,password);
```

The `tibemsConnection_Create` function exists for backward compatibility, but the recommended procedure is that you create `tibemsConnection` objects from factories.

See the `tibemsMsgProducer.c` sample client for a working example.

C#

Use the `ConnectionFactory.CreateConnection` method to create a `Connection` object:

```
Connection connection =  
    factory.CreateConnection(userName, password);
```

See the `csMsgProducer.cs` sample client for a working example.

Starting, Stopping and Closing a Connection

Before consuming messages, the Message Consumer client must "start" the connection. See [Creating a Message Consumer on page 346](#) for more details about Message Consumers.

If you wish to temporarily suspend message delivery, you can "stop" the connection.

When a client application exits, all open connections must be "closed." Unused open connections are eventually closed, but they do consume resources that could be used for other applications. Closing a connection also closes any sessions created by the connection.

See the "start," "stop" and "close" methods for the Java `Connection` object, the C `tibemsConnection` type, and the C# `Connection` object.

Creating a Session

A Session is a single-threaded context for producing or consuming messages. You create Message Producers or Message Consumers using Session objects. A Session can be transactional to enable a group of messages to be sent and received in a single transaction. A non-transactional Session can define the acknowledge mode of message objects received by the session. See [Message Acknowledgement on page 39](#) for details.

Java

Use the Connection object's `createSession()` method to create a Session object.

For example, to create a Session that uses the default `AUTO_ACKNOWLEDGE` session mode:

```
Session session = connection.createSession();
```

The EMS extended session modes, such as `NO_ACKNOWLEDGE`, require that you include the `com.tibco.tibjms.Tibjms` constant when you specify the EMS session mode. For example, to create a Session that uses the `NO_ACKNOWLEDGE` session mode:

```
Session session = connection.createSession(
    com.tibco.tibjms.Tibjms.NO_ACKNOWLEDGE);
```

See the `tibjmsMsgProducer.java` sample client for a working example.

C

Use the `tibemsConnection_CreateSession` function to create a session of type `tibemsSession`:

```
tibemsSession session = NULL;

status = tibemsConnection_CreateSession(connection,
    &session, TIBEMS_FALSE, TIBEMS_AUTO_ACKNOWLEDGE);
```

See the `tibemsMsgProducer.c` sample client for a working example.

C#

Use the `Connection.CreateSession` method to create a Session object:

```
Session session = connection.CreateSession(false,
    Session.AUTO_ACKNOWLEDGE);
```

See the `csMsgProducer.cs` sample client for a working example.

Setting an Exception Listener

All the APIs support the ability to set an exception listener on the connection that gets invoked when a connection breaks or experiences a fault-tolerant switchover.

When the event is a disconnect, the exception handler can call various EMS methods without any problem. However, when the event is a fault-tolerant switchover, the exception handler is not allowed to call any EMS method. To do so risks a deadlock. You can call the `setExceptionOnFTSwitch` method to receive an exception that contains the new server URL after a fault-tolerant switchover has occurred.

The following examples demonstrate how to establish an exception listener for a connection.

Java

Implement an `ExceptionListener.onException` method, use the `Connection` object's `setExceptionListener` method to register the exception listener, and call `Tibjms.setExceptionOnFTSwitch` to call the exception handler after a fault-tolerant switchover:

```
public class tibjmsMsgConsumer
    implements ExceptionListener
{
.....
    public void onException(JMSEException e)
    {
        /* Handle exception */
    }
.....
    connection.setExceptionListener(this);

    com.tibco.tibjms.Tibjms.setExceptionOnFTSwitch(true);
.....
}
```

See the `tibjmsMsgConsumer.java` sample client for a working example (without the `setExceptionOnFTSwitch` call).

C

Define an `onException` function to handle exceptions, use the `tibemsConnection_SetExceptionListener` function to call `onException` when an error is encountered, and call `tibems_setExceptionOnFTSwitch` to call the exception handler after a fault-tolerant switchover:

```
void onException(
    tibemsConnection conn,
    tibems_status     reason,
    void*             closure)
{
    /* Handle exception */
}

.....

status = tibemsConnection_SetExceptionListener(
                                connection,
                                onException,
                                NULL);

tibems_setExceptionOnFTSwitch(TIBEMS_TRUE);
```

See the `tibemsMsgConsumer.c` sample client for a working example (without the `setExceptionOnFTSwitch` call).

C#

Implement an `IExceptionListener.OnException` method, set the `Connection` object's `ExceptionListener` property to register the exception listener, and call `Tibems.SetExceptionOnFTSwitch` to call the exception handler after a fault-tolerant switchover:

```
public class csMsgConsumer : IExceptionListener
{
    .....
    public void OnException(EMSEException e)
    {
        /* Handle exception */
    }
    .....
    connection.ExceptionListener = this;

    TIBCO.EMS.Tibems.SetExceptionOnFTSwitch(true);
    .....
}
```

See the `csMsgConsumer.cs` sample client for a working example (without the `setExceptionOnFTSwitch` call).

Dynamically Creating Topics and Queues

EMS provides a JNDI implementation that can be used to store topics and queues. Java, C, and C# clients can use the EMS JNDI implementation to lookup topics and queues.

You can also store topics and queues in any JNDI-compliant naming service or in an LDAP server. Java clients can lookup topics and queues in any JNDI-compliant naming service. C and C# clients use LDAP servers.

[Looking up Administered Objects Stored in EMS on page 362](#) describes how to lookup topics and queues from an EMS server.

Clients can also create destinations as needed. If a client requests the creation of a destination that already exists, the existing destination is used. If the destination does not exist, and the specification of the `topics.conf`, `queues.conf`, or `acl.conf` files allow the destination, the server dynamically creates the new destination. The new destination inherits properties and permissions from its ancestors as described in [Wildcards and Dynamically Created Destinations on page 79](#). The destination is managed by the server as long as clients that use the destination are running.



Because dynamic destinations do not appear in the configuration files, a client cannot use JNDI to lookup dynamically created queues and topics.

The following examples show how to create destinations dynamically:

Java

Use the `Session` object's `createTopic()` method to create a topic as a `Destination` object:

```
Destination topic = session.createTopic(topicName);
```

Use the `Session` object's `createQueue()` method to create a queue as a `Destination` object:

```
Destination queue = session.createQueue(queueName);
```

See the `tibjmsMsgProducer.java` sample client for a working example.

C

Use the `tibemsTopic_Create` function to create a topic of type `tibemsDestination`:

```
tibemsDestination topic = NULL;
status = tibemsTopic_Create(&topic, topicName);
```

Use the `tibemsQueue_Create` function to create a queue of type `tibemsDestination`:

```
tibemsDestination queue = NULL;
status = tibemsQueue_Create(&queue, queueName);
```

See the `tibemsMsgProducer.c` sample client for a working example.

C#

Use the `Session.CreateTopic` method to create a `Topic` object:

```
Destination topic = session.CreateTopic(topicName);
```

Use the `Session.CreateQueue` method to create a `Queue` object:

```
Destination queue = session.CreateQueue(queueName);
```

See the `csMsgProducer.cs` sample client for a working example.

Creating a Message Producer

As described in [JMS Message Models on page 3](#), a *Message Producer* is an EMS client that either publishes messages to a topic or sends messages to a queue. When working with topics, a Message Producer is commonly referred to as a *Publisher*. Optionally, when creating a Message Producer, you can set the destination to NULL and specify the destination when you send or publish a message, as described in [Sending Messages on page 354](#).

You must have [send](#) permission on a queue to create a message producer that sends messages to that queue. You must have [durable](#) permission on the topic to create a new durable subscriber for that topic, and have at least [use_durable](#) permission on the topic to attach to an existing durable subscriber for the topic. See [User Permissions on page 283](#) for details.

The following examples create a message producer that sends messages to the queue that was dynamically created in [Dynamically Creating Topics and Queues on page 340](#).

Java

Use the Session object's `createProducer()` method to create a `MessageProducer` object:

```
MessageProducer QueueSender = session.createProducer(queue);
```

See the `tibjmsMsgProducer.java` sample client for a working example.

C

Use the `tibemsSession_CreateProducer` function to create a message producer of type `tibemsMsgProducer`:

```
tibemsMsgProducer QueueSender = NULL;
status = tibemsSession_CreateProducer(session,
                                     &QueueSender, queue);
```

See the `tibemsMsgProducer.c` sample client for a working example.

C#

Use the `Session.CreateProducer` method to create a `MessageProducer` object:

```
MessageProducer QueueSender = session.CreateProducer(queue);
```

See the `csMsgProducer.cs` sample client for a working example.

Configuring a Message Producer

A message producer can be configured to generate messages with default headers and properties that define how those messages are to be routed and delivered. Specifically, you can:

- Set the producer's default delivery mode.
- Set whether message IDs are disabled.
- Set whether message timestamps are disabled.
- Set the producer's default priority.
- Set the default length of time that a produced message should be retained by the message system.

For example, as described in the [Message Delivery Modes on page 25](#), you can set the message deliver mode to either `PERSISTENT`, `NON_PERSISTENT`, or `RELIABLE_DELIVERY`.

Java

Use the `MessageProducer` object's `setDeliveryMode()` method to configure your Message Producer with a default delivery mode of `RELIABLE_DELIVERY`:

```
QueueSender.setDeliveryMode(
    com.tibco.tibjms.Tibjms.RELIABLE_DELIVERY);
```

To configure the Message Producer with a default delivery mode of `NON_PERSISTENT`:

```
QueueSender.setDeliveryMode(
    javax.jms.DeliveryMode.NON_PERSISTENT);
```

See the `tibjmsMsgProducerPerf.java` sample client for a working example.



Delivery mode cannot be set by using the `Message.setJMSDeliveryMode()` method. According to the JMS specification, the publisher ignores the value of the `JMSDeliveryMode` header field when a message is being published.

C

Use the [tibemsMsgProducer_SetDeliveryMode](#) function to configure your Message Producer to set a default delivery mode for each message it produces to `RELIABLE_DELIVERY`:

```
tibems_int deliveryMode = TIBEMS_RELIABLE;
status tibemsMsgProducer_SetDeliveryMode(QueueSender,
                                          deliveryMode);
```

C#

Set the `DeliveryMode` on the `MessageProducer` object to `RELIABLE_DELIVERY`:

```
QueueSender.DeliveryMode = DeliveryMode.RELIABLE_DELIVERY;
```

See the `csMsgProducerPerf.cs` sample client for a working example.

Creating a Completion Listener for Asynchronous Sending

TIBCO Enterprise Message Service provides APIs for a Message Producer to send messages either synchronously or asynchronously. For asynchronous sending, you need to implement a `CompletionListener` that serves as an asynchronous event handler for message send result notification.

A completion listener implementation has two methods: `onCompletion()` is invoked after a message has successfully been sent, and `onException()` is invoked if the send failed. These methods are invoked in a different thread from that in which the message was sent. You implement the methods to perform the desired actions when the application is notified of send success or failure. Your implementation should handle all exceptions, and it should not throw any exceptions.

Once you create a completion listener, you pass it as an argument into the `MessageProducer` `send` method, or into the `JMSProducer` `setAsync()` method. If passed into the `JMSProducer` `setAsync` method, the `JMSProducer` will always send asynchronously.

Java

Create an implementation of the `CompletionListener` interface, create a `CompletionListener` and pass that into the appropriate send method:

```
/* create connection, session, producer, message */
TibjmsCompletionListener completionListener = new
    TibjmsCompletionListener();
msgProducer.send(destination, msg, completionListener);
```

Create a `CompletionListener` class and Implement the `onCompletion()` and `onException()` method to perform the desired actions when a message arrives: class `TibjmsCompletionListener` implements `CompletionListener`

```
{
    public void onCompletion(Message msg)
    {
        /* Handle the send success case for the message */
    }
}
```

```
public void onException(Message msg, Exception ex)
{
    /* Handle the send failure case for the message */
}
```

See the `tibjmsMsgProducer.java` sample client for a working example.

Creating a Message Consumer

Message consumers are clients that receive messages published to a topic or sent to a queue. When working with topics, a Message Consumer is commonly referred to as a *Subscriber*.

A Message Consumer can be created with a "message selector" that restricts the consumption of message to those with specific properties. When creating a Message Consumer for topics, you can set a `noLocal` attribute that prohibits the consumption of messages that are published over the same connection from which they are consumed.

Carefully consider the message selectors that are used with queue consumers. Because messages that do not match a queue consumer's message selectors remains in the queue until it is retrieved by another consumer, a non-matching message can experience many failed selectors. This is especially so when queue consumers connect, consume a message, and immediately disconnect.

As described in [Durable Subscribers for Topics on page 5](#), messages published to topics are only consumed by active subscribers to the topic; otherwise the messages are not consumed and cannot be retrieved later. You can create a durable subscriber that ensures messages published to a topic are received by the subscriber, even if it is not currently running. For queues, messages remain on the queue until they are either consumed by a Message Consumer, the message expiration time has been reached, or the maximum size of the queue is reached.

The following examples create a Message Consumer that consumes messages from the queue and a durable subscriber that consumes messages from a topic. The queue and topic are those that were dynamically created in [Dynamically Creating Topics and Queues on page 340](#).



The `createDurableSubscriber` method either creates a new durable subscriber for a topic or attaches the client to a previously created durable subscriber. A user must have `durable` permission on the topic to create a new durable subscriber for that topic. A user must have at least `use_durable` permission on the topic to attach to an existing durable subscriber for the topic. See [User Permissions on page 283](#) for details.

Java

Use the `Session` object's `createConsumer()` method to create a `MessageConsumer` object:

```
MessageConsumer QueueReceiver = session.createConsumer(queue);
```

See the `tibjmsMsgConsumer.java` sample client for a working example.

The following `Session.createDurableSubscriber()` method creates a durable subscriber, named "MyDurable":

```
TopicSubscriber subscriber =
    session.createDurableSubscriber(topic, "myDurable");
```

See the `tibjmsDurable.java` sample client for a working example.

Shared Subscriptions

Use the Session object's `createSharedConsumer()` method to create or add to a shared subscription:

```
MessageConsumer cons1 = session.createSharedConsumer(topic,
    "mySharedSub");
MessageConsumer cons2 = session.createSharedConsumer(topic,
    "mySharedSub");
```

`cons1` and `cons2` are two shared consumers on the same subscription called `mySharedSub`. If a message is published to the topic, then one of those two consumers will receive it. Note that shared consumers on a given subscription do not have to use the same session/connection.

Use the Session object's `createSharedDurableConsumer()` method to create or add to a shared durable subscription:

```
MessageConsumer cons1 = session.createSharedDurableConsumer(topic,
    "myDurableSharedSub");
MessageConsumer cons2 = session.createSharedDurableConsumer(topic,
    "myDurableSharedSub");
```

`cons1` and `cons2` are two shared durable consumers on the same durable subscription called `myDurableSharedSub`. If a message is published to the topic, then one of those two consumers will receive it. Note that shared durable consumers on a given subscription do not have to use the same session/connection.

C

Use the `tibemsSession_CreateConsumer` function to create a message consumer of type `tibemsMsgConsumer`:

```
tibemsMsgConsumer QueueReceiver = NULL;
status = tibemsSession_CreateConsumer(session,
    &QueueReceiver, queue, NULL, TIBEMS_FALSE);
```

See the `tibemsMsgConsumer.c` sample client for a working example.

The following `tibemsSession_CreateDurableSubscriber` function creates a durable subscriber, named "myDurable," of type `tibemsMsgConsumer`:

```
tibemsMsgConsumer msgConsumer = NULL;
status = tibemsSession_CreateDurableSubscriber(session,
    &msgConsumer, topic, "myDurable",
    NULL, TIBEMS_FALSE);
```

See the `tibemsDurable.c` sample client for a working example.

C#

Use the `Session.CreateConsumer` method to create a `MessageConsumer` object:

```
MessageConsumer QueueReceiver = session.createConsumer(queue);
```

See the `csMsgConsumer.cs` sample client for a working example.

The following `Session.CreateDurableSubscriber` method creates a durable subscriber, named "MyDurable":

```
TopicSubscriber subscriber =  
    session.CreateDurableSubscriber(topic, "myDurable");
```

See the `csDurable.cs` sample client for a working example.

Creating a Message Listener for Asynchronous Message Consumption

EMS allows a Message Consumer to consume messages either synchronously or asynchronously. For synchronous consumption, the Message Consumer explicitly calls a receive method on the topic or queue. For asynchronous consumption, you can implement a *Message Listener* that serves as an asynchronous event handler for messages.

A Message Listener implementation has one method, `onMessage`, that is called by the EMS server when a message arrives on a destination. You implement the `onMessage` method to perform the desired actions when a message arrives. Your implementation should handle all exceptions, and it should not throw any exceptions.

Once you create a Message Listener, you must register it with a specific Message Consumer before calling the connection's `start` method to begin receiving messages.

A Message Listener is not specific to the type of the destination. The same listener can obtain messages from a queue or a topic, depending upon the destination set for the Message Consumer with which the listener is registered.



The J2EE 1.3 platform introduced message-driven beans (MDBs) that are a special kind of Message Listener. See the J2EE documentation for more information about MDBs.

Java

Create an implementation of the `MessageListener` interface, create a `MessageConsumer`, and use the `MessageConsumer` object's `setMessageListener()` method to register the `Message Listener` with the `Message Consumer`:

```
public class tibjmsAsyncMsgConsumer implements MessageListener
{
    /* Create a connection, session and consumer */
    ...

    MessageConsumer QueueReceiver =
        session.createConsumer(queue);

    QueueReceiver.setMessageListener(this);

    connection.start();
}
```



Do not use the `Session.setMessageListener()` method, which is used by application servers, rather than by applications.

Implement the `onMessage()` method to perform the desired actions when a message arrives:

```
public void onMessage(Message message)
{
    /* Process message and handle exceptions */
}
```

See the `tibjmsAsyncMsgConsumer.java` sample client for a working example.

C

Implement an `onMessage()` function to perform the desired actions when a message arrives:

```
void onMessage(tibemsMsgConsumer QueueReceiver,
               tibemsMsg message, void* closure)
{
    /* Process message and handle exceptions */
}
```

In another function, that creates a `tibemsMsgConsumer` and uses the `tibemsMsgConsumer_SetMsgListener` function to create a message listener for the `Message Consumer`, specifying `onMessage()` as the callback function:

```
void run()
{
    tibemsMsgConsumer QueueReceiver = NULL;

    /* Create a connection, session and consumer */
    ...
}
```

```

status = tibemsSession_CreateConsumer(session,
                                     &QueueReceiver, queue, NULL, TIBEMS_FALSE);
status = tibemsMsgConsumer_SetMsgListener(QueueReceiver,
                                     onMessage, NULL);
status = tibemsConnection_Start(connection);
}

```

See the `tibemsAsyncMsgConsumer.c` sample client for a working example.

C#

Create an implementation of the `IMessageListener` interface, use `Session.CreateConsumer` to create a `MessageConsumer`, and set the `MessageListener` property on the `MessageConsumer` object to register the Message Listener with the Message Consumer:

```

public class csAsyncMsgConsumer : IMessageListener
{
    /* Create a connection, session and consumer */
    ...

    MessageConsumer QueueReceiver =
        session.CreateConsumer(queue);

    QueueReceiver.MessageListener = this;

    connection.Start();
}

```


Implement the `IMessageListener.OnMessage` method to perform the desired actions when a message arrives:

```
public void OnMessage(Message message) {  
    try  
    {  
        /* Process message and handle exceptions */  
    }  
}
```

See the `csAsyncMsgConsumer.cs` and `csAsyncMsgConsumerUsingDelegate.cs` sample clients for working examples.

Working with Messages

Messages are a self-contained units of information used by JMS applications to exchange data or request operations.

Creating Messages

As described in [JMS Message Bodies on page 22](#), EMS works with the following types of messages:

- Messages with no body
- Text Messages
- Map Messages
- Bytes Messages
- Stream Messages
- Object Messages

There is a separate create method for each type of message.

The following examples show how to create a simple text message containing the string "Hello."

Java

Use the `Session` object's `createTextMessage()` method to create a `TextMessage`:

```
TextMessage message = session.createTextMessage("Hello");
```

See the `tibjmsMsgProducer.java` sample client for a working example.

C

Use the `tibemsTextMsg_Create` function to create a text message of type `tibemsTextMsg`:

```
tibemsTextMsg message = "Hello";  
status = tibemsTextMsg_Create(&message);
```

See the `tibemsMsgProducer.c` sample client for a working example.

C#

Use the `Session.CreateTextMessage` method to create text message of type `TextMessage`:

```
TextMessage message = session.CreateTextMessage("Hello");
```

See the `csMsgProducer.cs` sample client for a working example.

Setting and Getting Message Properties

Before a client sends a message, it can use a "set property" method to set the message properties described in [EMS Message Properties on page 19](#). The client can check the message properties with a "get property" method.

Java

Use the `Message` object's `setBooleanProperty()` method to set the [JMS_TIBCO_PRESERVE_UNDELIVERED](#) property to true:

```
message.setBooleanProperty("JMS_TIBCO_PRESERVE_UNDELIVERED",
                           true);
```

Use the `getStringProperty()` method to get the user ID of the [JMS_TIBCO_SENDER](#):

```
userID = message.getStringProperty("JMS_TIBCO_SENDER");
```

C

Use the `tibemsMsg_SetBooleanProperty` function to set the [JMS_TIBCO_PRESERVE_UNDELIVERED](#) property to true:

```
status = tibemsMsg_SetBooleanProperty(message,
                                       "JMS_TIBCO_PRESERVE_UNDELIVERED", true);
```

Use the `tibemsMsg_GetStringProperty` function to get the user ID of the [JMS_TIBCO_SENDER](#):

```
char* userID = NULL;
status = tibemsMsg_GetStringProperty(message,
                                       "JMS_TIBCO_SENDER", &userID);
```

C#

Use the `Message.SetBooleanProperty` method to set the [JMS_TIBCO_PRESERVE_UNDELIVERED](#) property to true:

```
message.SetBooleanProperty("JMS_TIBCO_PRESERVE_UNDELIVERED",
                           true);
```

Use the `Message.GetStringProperty` method to get the user ID of the `JMS_TIBCO_SENDER`:

```
string userID = message.GetStringProperty("JMS_TIBCO_SENDER");
```

Sending Messages

You can use the Message Producer client, described in [Creating a Message Producer on page 342](#), to send messages to a destination. You can either send a message to the destination specified by the Message Producer or, if the Message Producer specifies NULL as the destination, you can send a message to a specific destination. In either case, you can optionally set the `JMSDeliveryMode`, `JMSExpiration`, and `JMSPriority` message header fields described in [JMS Message Header Fields on page 17](#) when sending each message.

The following examples show different ways to send a text message in each language:

- Send the message to the Message Producer, `QueueSender`, created in [Creating a Message Producer on page 342](#).
- Use a Message Producer with a NULL destination that sends the message to the topic created in [Dynamically Creating Topics and Queues on page 340](#).
- Use a Completion Listener, created in [Creating a Message Listener for Asynchronous Message Consumption on page 348](#), to send the message asynchronously.



Asynchronous sending is currently available only with the EMS client for Java.

See [Chapter 2, Messages](#) for more information about creating messages.

Java

Use the `MessageProducer` object's `send()` method to send a message to the destination specified by the `MessageProducer` object:

```
QueueSender.send(message);
```

Use the following form of the `send()` method to send a message to a specific destination:

```
MessageProducer NULLsender = session.createProducer(null);
....
NULLsender.send(topic, message);
```

Use the form of the `send()` method with a completion listener argument to send a message asynchronously:

```
QueueSender.send(message, completionListener);
```

See the `tibjmsMsgProducer.java` sample client for a working example.

C

Use the `tibemsMsgProducer_Send` function to send a message to the destination specified by the `tibemsMsgProducer`:

```
status = tibemsMsgProducer_Send(QueueSender, message);
```

Use the `tibemsMsgProducer_SendToDestination` function to send the message to a specific destination:

```
status = tibemsMsgProducer_SendToDestination(NULLsender,
                                              topic, message);
```

See the `tibemsMsgProducer.c` sample client for a working example.



Unlike the Java and C# APIs, in the C API, you can use the `tibemsMsgProducer_SendToDestination` function to specify the destination regardless of whether a destination is in the `tibemsMsgProducer`.

C#

Use the `MessageProducer.Send` method to send a message to the destination specified by the `MessageProducer`:

```
QueueSender.Send(message);
```

Use the following form of the `MessageProducer.Send` method to send a message to a specific destination:

```
MessageProducer NULLsender = session.CreateProducer(NULL);
NULLsender.Send(topic, message);
```

See the `csMsgProducer.cs` sample client for a working example.

Receiving Messages

The Message Consumer created in [Creating a Message Consumer on page 346](#) receives messages from a destination and acknowledges the receipt of messages using the mode established for the session, as described in [Creating a Session on page 337](#).

Before receiving messages, the Message Consumer must start the connection to the EMS server. Before exiting, the Message Consumer must close the connection.

The following examples start the connection created in [Connecting to the EMS Server on page 335](#); synchronously receive messages from the queue created in [Dynamically Creating Topics and Queues on page 340](#), and then close the connection.



You can also implement a Message Listener for your Message Consumer to asynchronously receive messages, as described in [Creating a Message Listener for Asynchronous Message Consumption on page 348](#).

Java

Use the Connection object's `start()` method to start the connection:

```
connection.start();
```

Use the MessageConsumer object's `receive()` method to receive a message. This is typically used in a loop for the duration the client wishes to receive messages:

```
Message message = QueueReceiver.receive();
```

When the client has finished receiving messages, it uses the `Close()` method to close the connection:

```
connection.close();
```

See the `tibjmsMsgConsumer.java` sample client for a working example.

C

Use the `tibemsConnection_Start` function to start the connection:

```
status = tibemsConnection_Start(connection);
```

Use the `tibemsMsgConsumer_Receive` function to receive a message. This is typically used in a loop for the duration the client wishes to receive messages:

```
tibemsMsg message = NULL;
```

```
status = tibemsMsgConsumer_Receive(QueueReceiver, &message);
```

When the client has finished receiving messages, use the `tibemsConnection_Close` function to close the connection:

```
status = tibemsConnection_Close(connection);
```

See the `tibemsMsgConsumer.c` sample client for a working example.

C#

Use the `Connection.Start` function to start the connection:

```
connection.Start();
```

Use the `MessageConsumer.Receive` function to receive a message. This is typically used in a loop for the duration the client wishes to receive messages:

```
Message message = QueueReceiver.receive();
```

When the client has finished receiving messages, use the `Connection.Close` function to close the connection:

```
connection.Close();
```

See the `csMsgConsumer.cs` sample client for a working example.

Chapter 12 Using the EMS Implementation of JNDI

The EMS server provides a implementation of JNDI that enables you to lookup connection factories, topics and queues, which are collectively referred to as *administered objects*. Java clients can look up administered objects stored in EMS using standard JNDI calls. The C and C# APIs provide similar calls to look up object data in the EMS server.

How to create topics and queues is described in [Creating and Modifying Destinations on page 75](#).

Topics

- [Creating and Modifying Administered Objects in EMS, page 360](#)
- [Looking up Administered Objects Stored in EMS, page 362](#)

Creating and Modifying Administered Objects in EMS

You can create administered objects for storage in EMS using either the administration tool or the administration APIs, or directly in the configuration files. This section describes how to create administered objects using the administration tool.

To create a connection factory, use the `create factory` command in the EMS Administration Tool. For example, to create a generic connection factory, named *myFactory*, that establishes a TCP connection to port 7344 on *server1*, start the EMS Administration Tool and enter:

```
create factory myFactory generic URL=tcp://server1:7344
```

The connection factory data stored on the EMS server is located in the `factories.conf` file. You can use the `show factories` command to list all of the connection factories on your EMS server and the `show factory` command to show the configuration details of a specific connection factory.

A connection factory may include optional properties for balancing server load and establishing thresholds for attempted connections, as described in [Connection Factory Parameters on page 245](#). These properties can be specified when creating the factory or modified for an existing factory using the `addprop factory`, `setprop factory`, and `removeprop factory` commands.

For example, to set the maximum number of connection attempts for the connection factory, *myFactory*, from the default value of 2 to 5, start the EMS Administration Tool and enter:

```
addprop factory myFactory connect_attempt_count=5
```

And to reset the value back to 2, enter:

```
setprop factory myFactory connect_attempt_count=2
```

Creating Connection Factories for Secure Connections

This section describes how to create a static connection factory for establishing an SSL connection. Similar SSL parameters must be used when looking up the connection factory, as described in [Performing Secure Lookups](#).

Connections that are to be secured using SSL identify the transport protocol as 'ssl' and may include any number of the SSL configuration parameters listed in [SSL Server Parameters on page 224](#).

For example, to create a generic connection factory, named *mySecureFactory*, that establishes a SSL connection to port 7243 on *server1*, start the EMS Administration Tool and enter:

```
create factory mySecureFactory generic URL=ssl://server1:7243
```

To create a factory to set up a generic connection and check the server's certificate to confirm the name of the server is `myServer`, enter (all one line):

```
create factory MySSLFactory generic url=ssl://7243
ssl_verify_host=enabled ssl_expected_hostname=myServer
ssl_trusted=certs/server_root.cert.pem
```

To create a factory to set up a topic connection, check the server's certificate (but not the name inside the certificate), and to set the `ssl_auth_only` parameter so that SSL is only used by the client when creating the connection, enter (all one line):

```
create factory AnotherSSLFactory topic url=ssl://7243
ssl_verify_host=enabled ssl_verify_hostname=disabled
ssl_trusted=certs/server_root.cert.pem ssl_auth_only=enabled
```



These samples assume that the certificate `server_root.cert.pem` is located in "certs" subdirectory of the directory where the server is running.

See [Chapter 18, Using the SSL Protocol](#), on page 465 for details.

Creating Connection Factories for Fault-Tolerant Connections

When connecting a fault-tolerant client to EMS, you must specify two or more EMS servers in your connection factory. When creating a connection factory for a fault-tolerant client, specify multiple server URLs in the `url` argument of the `create factory` command.

For example, to create a generic connection factory, named *myFtFactory*, that establishes TCP connections to port 7545 on the primary server, *server0*, and port 7344 on the backup server, *server1*, start the EMS Administration Tool and enter (on one line):

```
create factory myFtFactory generic url=tcp://server0:7545,
tcp://server1:7344
```

Should *server0* become unavailable, the client will connect to *server1*. See [Chapter 19, Fault Tolerance](#), on page 485 for details.

Looking up Administered Objects Stored in EMS

This section describes how to lookup objects from an EMS server by name.

All clients can lookup objects in the EMS naming service. Alternatively, Java applications can lookup objects in a third-party JNDI server, and C and C# clients can lookup objects in a third-party LDAP server.

To lookup administered objects stored in EMS, you need to create the initial context that identifies the URL of the naming service provider and any other properties, such as the username and password to authenticate the client to the service. The naming service provider URL has form:

```
tibjmsnaming://host:port
```

The following examples demonstrate how to access JMS administered objects when using TIBCO Enterprise Message Service. Each of these examples assume that a connection factory, named `ConFac`, exists in the `factories.conf` file, a topic `sample` exists in `topics.conf`, and a queue `sample` exists in `queues.conf`.

Java

Create an `InitialContext` object for the initial context, which consists of the provider context factory and JNDI provider URL, as well as the username and password to authenticate the client to the EMS server:

```
Hashtable env = new Hashtable();
env.put(Context.INITIAL_CONTEXT_FACTORY,
        "com.tibco.tibjms.naming.TibjmsInitialContextFactory");
env.put(Context.PROVIDER_URL, "tibjmsnaming://localhost:7222");
env.put(Context.SECURITY_PRINCIPAL, "userName");
env.put(Context.SECURITY_CREDENTIALS, "password");
InitialContext jndiContext = new InitialContext(env);
```

Look up a connection factory, named `ConFac`, and destinations, named `topic.sample` and `queue.sample`, from the initial context:

```
ConnectionFactory factory =
    (javax.jms.ConnectionFactory)
    jndiContext.lookup("ConFac");

javax.jms.Topic sampleTopic =
    (javax.jms.Topic)jndiContext.lookup("topic.sample");
javax.jms.Queue sampleQueue =
    (javax.jms.Queue)jndiContext.lookup("queue.sample");
```

See the `tibjmsJNDI.java` sample client located in the `EMS_HOME/samples/java/JNDI` directory.

C

Create a `tibemsLookupContext` object for the initial context, which consists of the JNDI provider URL and the username and password to authenticate the client to the EMS server:

```
tibemsLookupContext* contextstatus = NULL;
status = tibemsLookupContext_Create(
    &context,
    "tibjmsnaming://localhost:7222",
    "userName",
    "password");
```

Use the `tibemsLookupContext_LookupConnectionFactory` function to look up a connection factory, named `ConFac`, and use the `tibemsLookupContext_LookupDestination` function to look up the destinations, named `topic.sample` and `queue.sample`, from the initial context:

```
tibemsConnectionFactory factory = NULL;
tibemsDestination sampleTopic = NULL;
tibemsDestination sampleQueue = NULL;

status = tibemsLookupContext_Lookup(context,
    "ConFac",
    (void*)&factory);

status = tibemsLookupContext_Lookup(context,
    "sample.queue",
    (void*)&sampleQueue);

status = tibemsLookupContext_Lookup(context,
    "topic.sample",
    (void*)&sampleTopic);
```

C#

Create a `ILookupContext` object for the initial context, which consists of the JNDI provider URL and the username and password to authenticate the client to the EMS server:

```
Hashtable env = new Hashtable();
env.Add(LookupContext.PROVIDER_URL,
    "tibjmsnaming://localhost:7222");
env.Add(LookupContext.SECURITY_PRINCIPAL, "myUserName");
env.Add(LookupContext.SECURITY_CREDENTIALS, "myPassword");

LookupContextFactory factory = new LookupContextFactory();
```

```
ILookupContext searcher = factory.CreateContext(
    LookupContextFactory.TIBJMS_NAMING_CONT
    EXT,
    env);
```

Use the `ILookupContext.Lookup` method to look up a connection factory, named `ConFac`, and destinations, named `topic.sample` and `queue.sample`, from the initial context:

```
ConnectionFactory factory =
    (ConnectionFactory) searcher.Lookup("ConFac");

Topic sampleTopic =
    (Topic)searcher.Lookup("topic.sample");

TIBCO.EMS.Queue sampleQueue =
    (TIBCO.EMS.Queue)searcher.Lookup("queue.sample");
```

Looking Up Objects Using Full URL Names

Java clients can look up administered objects using full URL names. In this case, the `Context.URL_PKG_PREFIXES` property is used in place of the `Context.PROVIDER_URL` property. For example:

```
Hashtable env = new Hashtable();
env.put(Context.URL_PKG_PREFIXES, "com.tibco.tibjms.naming");
env.put(Context.PROVIDER_URL, "tibjmsnaming://localhost:7222");
env.put(Context.SECURITY_PRINCIPAL, "userName");
env.put(Context.SECURITY_CREDENTIALS, "password");
jndiContext = new InitialContext(env);
```

When using full URL names, you can look up objects like the following example:

```
Topic sampleTopic = (javax.jms.Topic)jndiContext.lookup(
    "tibjmsnaming://jmshost:7222/topic.sample");
Queue sampleQueue = (javax.jms.Queue)jndiContext.lookup(
    "tibjmsnaming://jmshost:7222/queue.sample");
```

For further information on how to use full URL names, refer to the `tibjmsJNDIRead.java` example located in the `EMS_HOME/samples/java/JNDI` directory.

Performing Secure Lookups

TIBCO Enterprise Message Service client programs can perform secure JNDI lookups using the Secure Sockets Layer (SSL) protocol. To accomplish this, the client program must set SSL properties in the environment when the `InitialContext` is created. The SSL properties are similar to the SSL properties for the TIBCO Enterprise Message Service server. See [Chapter 18, Using the SSL Protocol](#) for more information about using SSL in the TIBCO Enterprise Message Service server.

The following examples illustrate how to create an `InitialContext` that can be used to perform JNDI lookups using the SSL protocol.

Java

In this example, the port number specified for the `Context.PROVIDER_URL` is set to the SSL listen port that was specified in the server configuration file `tibjsmd.conf`. The value for `TibjmsContext.SECURITY_PROTOCOL` is set to `ssl`. Finally, the value of `TibjmsContext.SSL_ENABLE_VERIFY_HOST` is set to `"false"` to turn off server authentication. Because of this, no trusted certificates need to be provided and the client will then not verify the server it is using for the JNDI lookup against the server's certificate.

```
Hashtable env = new Hashtable();
env.put(Context.INITIAL_CONTEXT_FACTORY,
        "com.tibco.tibjms.naming.TibjmsInitialContextFactory");
env.put(Context.PROVIDER_URL, tibjmsnaming://jms host:7223);
env.put(Context.URL_PKG_PREFIXES, "com.tibco.tibjms.naming");
env.put(TibjmsContext.SECURITY_PROTOCOL, "ssl");
env.put(TibjmsContext.SSL_ENABLE_VERIFY_HOST,
        new Boolean("false"));
Context context = new InitialContext(env);
```

C

Create a `tibemsSSLParams` object and use the `tibemsSSLParams_SetIdentityFile` function to establish the client identity by means of a `pkcs12` file. Use the `tibemsLookupContext_CreateSSL` function to create a `tibemsLookupContext` object that uses an SSL connection for the initial context.

```
tibemsLookupContext*    context    = NULL;
tibemsConnection_Factory factory    = NULL;
tibemsSSLParams          sslParams  = NULL;
tibems_status            status     = TIBEMS_OK;

sslParams = tibemsSSLParams_Create();
```

```

status = tibemsSSLParams_SetIdentityFile(
    ssl_params,
    "client_identity.p12",
    TIBEMS_SSL_ENCODING_AUTO);

status = tibemsLookupContext_CreateSSL(
    &context,
    "tibjmsnaming://localhost:7222",
    "userName",
    "password",
    sslParams,
    "pk_password");

```

C#

Create a `ILookupContext` object for the initial context over an SSL connection. The SSL Store Info consists of a `pkcs12` file that identifies the client and the client's password, which are stored in an `EMSSSLFileStoreInfo` object.

```

string ssl_identity = client_identity.p12;
string ssl_target_hostname = "server";
string ssl_password = "password";

EMSSSLFileStoreInfo StoreInfo = new EMSSSLFileStoreInfo();
info.SetSSLClientIdentity(ssl_identity);
info.SetSSLPassword(ssl_password.ToCharArray());

Hashtable env = new Hashtable();
env.Add(LookupContext.PROVIDER_URL, "adc1.na.tibco.com:10636");
env.Add(LookupContext.SECURITY_PRINCIPAL, "myUserName");
env.Add(LookupContext.SECURITY_CREDENTIALS, "myPassword");
env.Add(LookupContext.SECURITY_PROTOCOL, "ssl");
env.Add(LookupContext.SSL_TARGET_HOST_NAME,
    ssl_target_hostname);
env.Add(LookupContext.SSL_STORE_TYPE,
    EMSSSLStoreType.EMSSSL_STORE_TYPE_FILE);
env.Add(LookupContext.SSL_STORE_INFO, StoreInfo);

```

Performing Fault-Tolerant Lookups

TIBCO Enterprise Message Service can perform fault-tolerant JNDI lookups. If the primary server fails and the backup server becomes the primary, the JNDI provider automatically uses the new primary server for JNDI lookups. You accomplish this by providing multiple URLs in the `Context.PROVIDER_URL` property when creating the `InitialContext`. Specify more than one URL separated by commas (,) in the property.

Example

The following illustrates setting up the `Context.PROVIDER_URL` property with the URLs of a primary EMS server on the machine named `emshost` and a backup EMS server on the machine named `backuphost`.

```
env.put(Context.PROVIDER_URL, "tibjmsnaming://jmshost:7222,  
tibjmsnaming://backuphost:7222");
```

If at any time the first EMS server fails, the JNDI provider will automatically switch to the EMS server on the host `backuphost` for JNDI lookups. If `emshost` is repaired and restarted, it then becomes the backup EMS server.

Limitations of Fault-Tolerant JNDI Lookups

Fault-tolerant JNDI lookups do not occur in the following scenarios:

- When using full URL names in argument to the lookup method.
- When looking up an object that has been bound into a foreign naming/directory service such as LDAP.

Chapter 13 **Using Multicast**

Multicast is a messaging model that allows the EMS server to send messages to multiple consumers simultaneously by broadcasting them over an existing network. This chapter describes how to use and configure multicast in TIBCO Enterprise Message Service.

Topics

- [Overview of Multicast, page 370](#)
- [Configuring Multicast, page 374](#)
- [Running Multicast, page 379](#)
- [Monitoring and Statistics, page 380](#)

Overview of Multicast

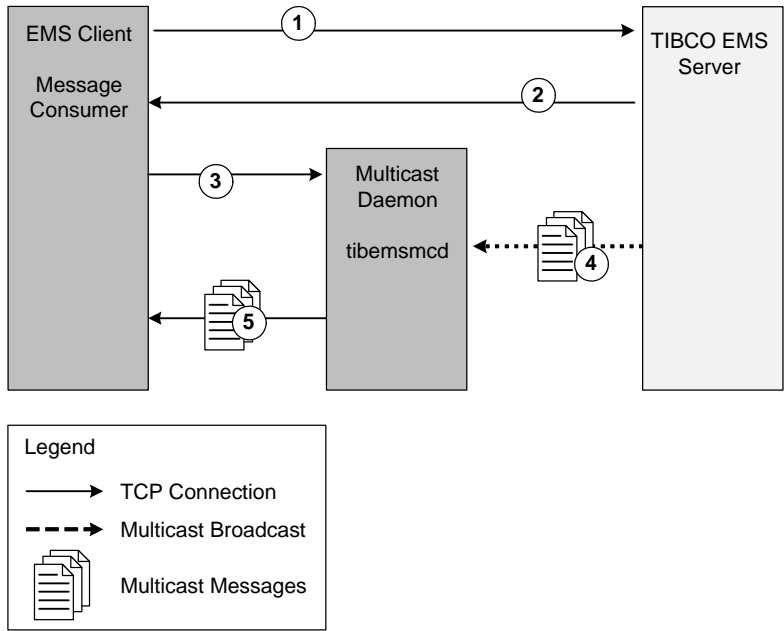
Multicast is a messaging model that broadcasts messages to many consumers at once, as opposed to sending copies of a message to each subscribing consumer individually. TIBCO Enterprise Message Service uses Pragmatic General Multicast (PGM) to broadcast messages published to multicast-enabled topics over an existing network. Messages sent to topics that are not multicast-enabled are delivered to the message consumer using TCP.

The server sends multicast messages over a *multicast channel*. Each multicast-enabled topic is associated with a channel. The channel determines the multicast port and multicast group address to which the server sends messages.

The multicast message is received by a *multicast daemon* running on the same computer with the message consumer. When an EMS client subscribes to a multicast-enabled topic, it automatically connects to the multicast daemon. The multicast daemon begins listening on the channel associated with that topic, receives any broadcast messages, and delivers them to subscribed clients.

Figure 18 shows the communication flow between a multicast message consumer, EMS server, and multicast daemon.

Figure 18 Multicast message consumer creation



The following describes the multicast message consumer creation process:

1. The EMS client connects to the EMS server and subscribes to one or more multicast-enabled topics.
2. The EMS server sends a reply to the client, including instructions and configuration information for the multicast daemon.
3. The client connects to the multicast daemon and passes the configuration information from the server. The multicast daemon then begins listening for multicast messages from the server.
4. The server begins broadcasting messages, which the multicast daemon receives.
5. The multicast daemon delivers the messages to the client.

The client will continue to receive non-multicast messages directly from the server.

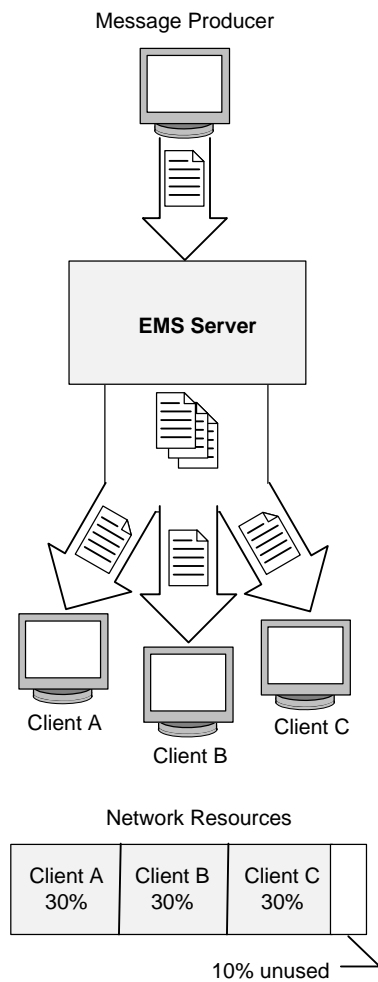
When to Use Multicast

Because multicast reduces the number of operations performed by the server and reduces the amount of bandwidth used in the publish and subscribe model, multicast is highly scalable.

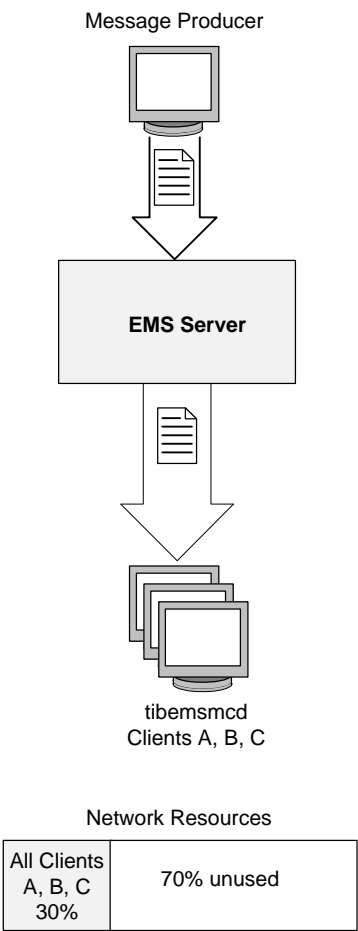
[Figure 19 on page 372](#) shows how using multicast can reduce the amount of bandwidth used to send a message. Where publish and subscribe messaging creates a copy of a published message for each message consumer, multicast broadcasts the message only once. Multiple multicast daemons listening on the channel receive the same broadcast.

Figure 19 The benefits of multicast

Point-to-Point Messaging



Multicast Messaging



Although multicast can reduce the network resources used by the server, it is not the best messaging model for every system. Multicast offers last-hop delivery only; it cannot be used to send messages between servers. However, messages sent to multicast-enabled topics are still delivered to other subscribed servers using the standard TCP connection.



Multicast does not guarantee message delivery. Messages requiring a high degree of reliability should not use multicast.



The multicast daemon and message consumer always reside on the same machine, and PGM is used to deliver the broadcast message from EMS server to the daemon. Because there is no security on PGM, multicast should not be used in applications where security is a priority.

Requirements

In order to use multicast in your EMS messaging application, the following requirements must be met:

- The EMS server must be configured for multicast:
 - The server must be enabled for multicast.
 - Multicast channels must be configured.
 - The desired topics must be multicast-enabled by associating them with a multicast channel. Multicast is not compatible with queues.

See [Configuring Multicast on page 374](#) for more information about configuring multicast.

- The multicast-enabled message consumer must be created using the `NO_ACKNOWLEDGE` mode. See [Message Acknowledgement on page 39](#) for more information.
- The multicast daemon must be running on the same computer as the subscriber. See [Starting the Multicast Daemon on page 379](#) for more information.

Backwards Compatibility

Multicast is backwards compatible, and can be used in applications where not all EMS clients are using the same version of TIBCO Enterprise Message Service. The EMS sever and any clients that wish to receive messages over multicast must be Software Release 5.0 or later.

Multicast is configured primarily in the EMS server, and is largely invisible to EMS clients. Message producers do not need to be enabled for multicast in order to send multicast messages, because topics are multicast-enabled in the server. However, clients are multicast-enabled by default.

Clients that are not multicast-enabled, either because multicast has been disabled or because the client uses a release of EMS earlier than 5.0, will receive messages from the server over the TCP connection, even if the message topic is multicast-enabled.

Configuring Multicast

Multicast is configured in the EMS server configuration files. Configuration is a simple three-step process:

1. Enable multicast in the EMS server.

Enable the `multicast` parameter in the `tibemsd.conf` file. Optional multicast parameters allow you to control other settings, such as the default multicast daemon port and the maximum amount of multicast traffic allowed. See [Multicast Parameters on page 217](#) for more information.

2. Create multicast channels.

Create named channels in the `channels.conf` file. See [channels.conf on page 241](#) for more information about the channels configuration file.

3. Associate topics with channels.

In the `topics.conf` configuration file, add the `channel` property to the definitions of those topics you wish to be multicast. See [channel on page 59](#) for more information about the channel property. Note that a topic can be associated with only one multicast channel.

Configuring Multicast Dynamically

For the most part, multicast is configured statically. Only limited changes can be made to multicast settings during runtime. Once the EMS server has been started, the only multicast configuration change you can make is to the `channel` property of a topic. With the administration tool, you can assign to or remove an assigned multicast channel from a topic. You cannot change the channel configuration or the `channels.conf` file. To do that, you must stop the server.

These commands can be used to change a topic's `channel` property:

- `addprop topic` adds the `channel` property to a topic. For example, this sets the `channel` property for the topic `foo.bar` to `mychannel`:

```
addprop topic foo.bar channel=mychannel
```

However, although this enables the topic `foo.bar` for multicast, current subscribers to the topic will continue to receive messages over TCP. An existing message consumer will not receive messages sent to `foo.bar` over multicast until the consumer has been stopped and restarted.

- `setprop topic` offers the same functionality as `addprop topic`, with one important difference: when `setprop topics` is used, it resets all other properties to their default values.

This command also enables the topic for multicast, but does not cause existing topic subscribers to receive messages over multicast. Only messages consumers that are created after the `channel` property is set will receive multicast messages.

- `removeprop topic` removes the `channel` property from the topic. Current multicast subscribers will begin to receive messages sent to the topic over TCP.

If a backlog of messages exists in the server or multicast daemon, the EMS client may receive some messages out of order, and some message loss is possible. The multicast daemon will continue to deliver queued messages until the backlog is gone, while the EMS server will deliver later messages immediately.



A current topic subscriber will stop receiving messages if the multicast channel is changed from one channel to a different channel. This can happen when:

- The channel is changed explicitly using `addprop topic` or `setprop topic`.
- The channel is removed using `removeprop topic`, and the topic inherits a different channel from a parent.

If the channel assigned to a topic changes, current subscribers to the topic will not receive messages until they have resubscribed to the topic. The server will *not* send messages to the client over TCP if there is another channel assigned to the topic.

In general, we recommend changing channels only when subscribers are stopped.

Configuring the Multicast Daemon

The multicast daemon, or `tibemsmcd`, is the process that receives multicast messages from EMS servers and delivers them to individual clients. The multicast daemon runs on the local host computer with the client. One daemon can receive messages from multiple servers, and can deliver messages to multiple clients.

Configuration for the multicast daemon is set in the EMS server, and passed to the daemon when the EMS client creates a multicast message consumer and connects to the multicast daemon. In some cases, you may wish to make configuration changes to the daemon directly. You can do this using command line options.

For example, if your configuration requires more than one network interface on a single computer, you can run multiple multicast daemons on the local host. Use the `-ifc` command line option to change the interface for a daemon. See [Command Line Options](#) below for more information.

Command Line Options

The multicast daemon accepts a few command-line options. When starting `tibemsmcd`, you can specify the following options:

Table 61 `tibemsmcd` Options

Option	Description
<code>-ifc interface</code>	<p>Select the IP address that identifies the network interface used by the multicast daemon to receive multicast data.</p> <p>If this option is not included, the multicast daemon uses the default interface, determined by the IP address <code>INADDR_ANY</code>.</p> <p>If your configuration requires multiple interfaces, you will need one multicast daemon instance for each interface. It may be helpful to use the commands <code>ipconfig</code> on Windows or <code>ifconfig</code> on UNIX systems to determine what interfaces are available and support multicast.</p>
<code>-help</code> or <code>-h</code>	Print the help screen.
<code>-logfile file</code>	Specify a logfile where trace messages will be written.
<code>-logfile-max-size size[KB MB GB]</code>	<p>Specify the maximum size that the trace message log file may reach before rotation. By default, log files have no size limit and so do not rotate.</p> <p>Zero is a special value, which specifies no maximum size. Otherwise, the value you specify must be greater than or equal to 64KB.</p>

Table 61 *tibemsmcd* Options

Option	Description
<code>-listen [ip-address:]tcp-port</code>	<p>Change the IP address and TCP port on which the daemon listens for connections from EMS clients, where:</p> <ul style="list-style-type: none"> <i>ip-address</i> is an optional parameter that, when provided, restricts the interface on which the multicast daemon will accept client connections to a specific IP address. If an <i>ip-address</i> is not provided, the multicast daemon listens for EMS clients on all interfaces. <i>tcp-port</i> is the TCP port on which the daemon listens for connections from EMS clients. The default port is 7444. <p>For example:</p> <pre>-listen 127.0.0.1:7444.</pre> <p>Note that if the default TCP port that the daemon listens on is changed, then the client must be directed to attempt a connection to the daemon on the same TCP port. To change the port that the client uses, set the <code>multicast_daemon_default</code> parameter in the <code>tibemsd.conf</code> file.</p>
<code>-trace</code>	<p>Enable tracing in the multicast daemon. If this option is included, trace information for events such as client connections to the daemon and channel creation is written to file.</p>
<code>-max-msg-memory size[MB KB]</code>	<p>Specify the maximum amount of memory allowed for all messages waiting to be sent to consumers. Once the specified memory limit is reached, new messages are discarded. Specify the size in units of MB or GB. The minimum permitted size is 8MB.</p>
<code>-max-loss-rate percentage</code>	<p>Specify the maximum percentage (between 1 and 100) of acceptable loss rate. When the rate rises above the given percentage, the multicast daemon stops sending NAKs.</p> <p>By default, the maximum loss rate is 10%.</p>
<code>-no-console-trace</code>	<p>Prevent the multicast daemon from sending tracing messages to the console.</p>

Controlling Access to Multicast-Enabled Topics

Publish and subscribe permissions for multicast-enabled topics are controlled the same way that they are controlled for topics that are not multicast-enabled. See [Destination Control on page 276](#) for more information about controlling access to destinations.

Running Multicast

For an example of multicast messaging, see [Multicast Messaging Example on page 104](#)

Starting the Multicast Daemon

The multicast daemon is located in your *installation_path* /bin directory and is a stand-alone executable named `tibemsmcd` on UNIX and `tibemsmcd.exe` on Windows platforms.



On a computer running Windows, you can also start the multicast daemon from the Start menu, following the path **Programs > TIBCO > TIBCO EMS 8.0 > Start EMS Multicast Daemon**.

Creating a Multicast Consumer

The EMS client is enabled for multicast by default, and no special configuration is required. To receive multicast data, the client need only create a multicast consumer by subscribing to a multicast-enabled topic using the `NO_ACKNOWLEDGE` mode, as described in [Message Acknowledgement on page 39](#).

You can also disable multicast in a client, using API calls. For more information, see the API documentation for your language.

Monitoring and Statistics

There are a number of aspects of a multicast deployment that can be analyzed to determine the deployment's health and status and to aid in troubleshooting.

Monitoring

The server publishes messages to two monitoring topics specifically related to multicast. These topics are `$sys.monitor.multicast.status` and `$sys.monitor.multicast.stats`.

The server publishes monitoring messages to the topic `$sys.monitor.multicast.status`. These messages contain information about the status of a multicast consumer and the multicast daemon to which it is connected. This information includes when a consumer has successfully joined a multicast group and when a consumer experiences an error, such as unrecoverable loss in its multicast daemon. By monitoring multicast errors you can detect which consumers are experiencing problems, allowing you to take corrective action.

Low-level multicast statistics are published in a monitoring message to the topic `$sys.monitor.multicast.stats`. The statistics include information such as the number of bytes sent to a multicast group and the number of NAKs sent by a multicast daemon. These multicast statistics can aid in troubleshooting a multicast deployment when provided to TIBCO technical support. Generally these statistics won't have much meaning to a typical user. Multicast statistics are only published when the server's `multicast_statistics_interval` is set to a non-zero value. By default the `multicast_statistics_interval` is set to zero.

See [Chapter 17, Monitoring Server Activity](#) for more information on monitoring.

Statistics

The server's multicast channel statistics can be viewed using the administration API or the administration command line tool. Multicast channel statistics include:

- The average number of messages sent per second.
- The average number of bytes sent per second.
- The total number of messages sent.
- The total number of bytes sent.
- Detailed statistics for each topic using the channel.

See [Working with Server Statistics on page 460](#) for more information on statistics.

Chapter 14 **Multicast Deployment and Troubleshooting**

This chapter reviews important multicast deployment considerations, and provides hints and suggestions for countering some common problems associated with multicast deployments.

Topics

- [Deployment Considerations, page 382](#)
- [Walking Through a Multicast Deployment, page 388](#)
- [Troubleshooting EMS Multicast, page 397](#)

Deployment Considerations

Ensuring a proper multicast deployment takes some forethought, more than a traditional unicast deployment. This section discusses some subjects to consider before deploying TIBCO Enterprise Message Service with multicast.

Issues in multicast deployment can be separated into three areas: ensuring multicast connectivity, restricting multicast traffic, and managing bandwidth. These can be represented with three basic questions:

1. Can multicast traffic go to all hosts where it is wanted? See [Connectivity on page 382](#).
2. Will multicast traffic go to any hosts where it is unwanted? See [Restricting Multicast Traffic on page 384](#).
3. How will unicast and multicast traffic share the available network bandwidth? See [Managing Bandwidth on page 384](#).

Connectivity

Like unicast applications, multicast applications require that the network layer provide a path for multicast data to flow from senders to receivers. However, routers and switches may require additional configuration for multicast use and tuning. The first step in ensuring and limiting connectivity is defining channels, and assigning multicast group addresses these channels.

Multicast Addresses

Each multicast channel, defined in the `channels.conf` configuration file, is assigned a multicast address. TIBCO Enterprise Message Service allows you to assign any valid multicast address, in the class D address range, 224.0.0.0 through 239.255.255.255. However, in order to avoid a conflict, please refer to the Internet Assigned Numbers Authority (IANA) list of reserved addresses to avoid a conflict:

<http://www.iana.org/assignments/multicast-addresses>

When assigning addresses to your channels, keep these additional considerations in mind:

- Multicast addresses 224.0.1.78 and 224.0.1.79 are reserved by TIBCO EMS for internal use. These addresses should not be used, as TIBCO multicast traffic may be encountered there.
- Ideally, you should select multicast addresses from 239.0.0.0 to 239.255.255.255. These have been set aside as an administratively scoped

block, and IANA will never reserve these. They can be freely used within your enterprise without worry of any external conflict.

- There is not a one-to-one mapping of MAC addresses to IP addresses; because of this you should not pick x.0.0.x addresses, as they may map to reserved addresses and so may not work. The class D IP address range assigned to multicasting is 28 bits wide, but the range of MAC addresses assigned to multicast is only 23 bits wide. Since only the 23 lower order bits of the IP address are assigned to make the MAC address, an overlap results. For example, if one chooses a multicast address 239.0.0.1, it may incorrectly overlap to the reserved 224.0.0.1.

Defining Channels

TIBCO Enterprise Message Service does not restrict the number of channels that you can configure and use in the EMS server or the multicast daemon. However, the number of IP multicast group addresses that can be joined by any one host at one time may be constrained by outside factors. Often, the number is limited by the NIC, and typically this limitation is not specified in the NIC documentation.

Experimentation is often the only way to determine what the limit is for a specific NIC and OS. With some NICs, joining too many groups will set the card to "promiscuous mode" which will adversely affect performance.

It is also important to note that, because a channel represents both an IP multicast group address *and* a destination port, there is not necessarily a one-to-one correlation between a channel and multicast group.

A group is joined when a multicast daemon listens to an IP multicast group address. Because a channel represents both an IP multicast group address *and* a destination port, there is not necessarily a one-to-one correlation between a channel and multicast group. For example, if you have 10 multicast channels all using the same multicast group address but different ports, then a multicast daemon will join at most one group. However, if the 10 multicast channels are all using different multicast group addresses, then a multicast daemon may join up to 10 groups.

The multicast IP address and port combinations that you choose should only be used with TIBCO EMS. While the TIBCO Multicast Daemon can filter out corrupt network data, receiving data packets that are not specific to EMS can yield unpredictable results, which could destabilize your network.

Ensuring Multicast connectivity

As stated earlier, multicast applications require that the network layer provide a path for multicast data to flow from senders to receivers. By default, most routers and switches have multicast routing disabled and require additional configuration to enable it. If you experience connectivity problems, this is the first place to check.

For example, with CISCO routers you must use the `ip multicast-routing` command to enable multicast routing. Multicast hardware configuration falls outside the scope of this document; please consult your network administrator or the TIBCO Professional Services Group for configuration specific to your network and enterprise.

Restricting Multicast Traffic

Multicast deployment often also involves making sure that multicast streams do not go where they are unwanted, especially when high-bandwidth streams are present on a network that also includes some low-bandwidth links, or where access must be controlled at the network layer for security reasons.

Within a LAN, Ethernet switches can direct unicast traffic only to ports where it is wanted. Typically, because routers and switches do not enable multicast packet forwarding by default, restricting multicast traffic is not an issue. However, one must be cognizant of this issue when planning a multicast deployment.

Managing Bandwidth

This section discusses bandwidth considerations that are specific to multicast deployments. There are three main aspects to bandwidth:

- [Determining Available Bandwidth, page 385](#) — determine your available bandwidth, and setting bandwidth limitations to maximize performance.
- [Dividing Bandwidth Among Channels, page 386](#) — create channels to make the best use of available bandwidth.
- [Handling Slow Applications, page 387](#) — managing small numbers of slow applications so that they do not slow the entire multicast network.

Determining Available Bandwidth

Reliable unicast transports, such as TCP, automatically share available network bandwidth among all sessions contending for it. Administrators play no role in this process; the available bandwidth is dynamically determined by the protocol stacks as they measure the round-trip time and packet loss rates. This process is called *congestion control*. It assumes that all streams have equal priority and it automatically divides bandwidth accordingly.

In contrast, multicast relies on the administrator to ensure that the amount of bandwidth the network delivers is reserved or available. In TIBCO Enterprise Message Service, the administrator allocates network bandwidth for each multicast channel using the `maxrate` configuration parameter (see [channels.conf on page 241](#)). Correctly allocating bandwidth prevents the application from experiencing congestion.

Congestion can cause packet loss, which can in turn cause erratic behavior or even application failure. This is another significant difference between multicast and unicast; with unicast, congestion causes applications to run more slowly, but will not cause them to fail.

You must carefully consider and limit how fast you send, because TIBCO Enterprise Message Service does not impose bandwidth limitations. If you try to send faster than the network can actually deliver the data, you will see *substantially* lower throughput than had you asked for slightly less bandwidth than the network can actually deliver.

It is somewhat paradoxical, but if you ask the EMS server to deliver 900 Mbps over a network layer that can deliver 1 Gbps, it will. If you ask it to deliver more than 1 Gbps over a 1 Gbps network layer, you could get as little as 400 Mbps. What will most likely occur is chaotic behavior based on loss rates and other factors.

This leads to an unusual rule: *if throughput is too low, try asking for less—there is a chance you may get more*. It is important to perform this test even if your throughput is still well below "wire speed." That is because loss due to congestion can come from many sources other than the wire speed limit, such as TCP data on the same network. It is a simple test and if the results show that actual throughput goes up as the amount of bandwidth requested goes down, it is a very strong sign that there is loss due to congestion somewhere in your network, between the sender and receivers.



Restrict multicast traffic to a rate a little *below* the maximum capacity of your network. If your throughput rate is slower than expected, restrict the rate further. You may find that throughput actually increases.

To set the rate for multicast traffic on a channel, see the `maxrate` parameter, in [channels.conf on page 241](#).

You can think of the bandwidth rate specified for a channel as a delivery promise that the network layer makes to EMS. If the network layer breaks that promise, EMS multicast throughput falls to a rate substantially below what the network can actually deliver.

Dividing Bandwidth Among Channels

Ideally, a deployment within a set of routed subnets, or VLAN, should have hosts with heterogeneous interfaces of homogeneous speed. Deployments that do not adhere to this are not recommended, because loss can be introduced if the receiving interfaces are slower than the link and sending interface. This happens because the slower interfaces cannot handle bursts of data on a faster network. Also, we do not recommend that you use EMS multicast over WAN links.



Following these recommendations will help minimize data loss due to bandwidth inconsistencies:

- Multicast publishers and subscribers should have network interfaces of the same speed.
- The ideal multicast deployment is over a LAN or VLAN.

For example, if you have a number of clients with 100Mb NIC cards and others with 1Gb NIC cards, the recommended architecture is to send from a 100Mb NIC to the slower receivers and a 1Gb NIC to the faster receivers. You can accomplish this by configuring two multicast channels, one for the faster-speed senders and receivers, and one for the slower senders and receivers.

Alternatively, you can configure one channel and limit the bandwidth to the slowest receiver, or 100Mb. However, the best solution is to use a multi-homed machine, separate the applications by defining different channels for two interfaces, then allowing each channel to operate at its optimum speed.

For example, these two channel configurations are optimized for 100Mb NIC card and a 1Gb NIC card:

```
--- channels.conf ---
[channel_100mb]
  address = 239.1.1.1:10
  maxrate = 7MB
  interface=10.99.99.99

[channel_1Gb]
  Address = 239.1.1.2:10
  maxrate = 95MB
  interface=10.99.99.100
```

Applications running on 100Mb machines would use topics with `channel_100Mb` assigned to them, and applications on machines with 1 Gb NIC cards would use topics with `channel_1Gb` assigned. Also note that some bandwidth has been left for other TCP data, as suggested in [Determining Available Bandwidth on page 385](#).

Handling Slow Applications

If you have a small number of applications or hosts that are known to be "slow" or are on a WAN, but need to subscribe to the data on a multicast enabled topic, we recommend disabling EMS multicast at the application. You can disable multicast in a client through API calls; see the API documentation for your language.

The slow application will receive messages from the server over TCP, effectively removing them from the multicast stream and avoiding congesting and slowing down other multicast receivers. It is very important to account for the TCP bandwidth used by application(s) that do this in your multicast bandwidth calculations.



If an EMS client with multicast disabled subscribes to a topic that is multicast-enabled, messages will be delivered to the client over TCP. Take this TCP traffic into consideration when setting your bandwidth limitations, as described in [Determining Available Bandwidth on page 385](#).

Walking Through a Multicast Deployment

This section describes the steps needed to set up a simple example TIBCO Enterprise Message Service multicast deployment:

- [Step 1: Design the Multicast Network Architecture, page 388](#)
- [Step 2: Install and Set Up EMS, page 390](#)
- [Step 3: Determine Network and Application Capabilities, page 393](#)

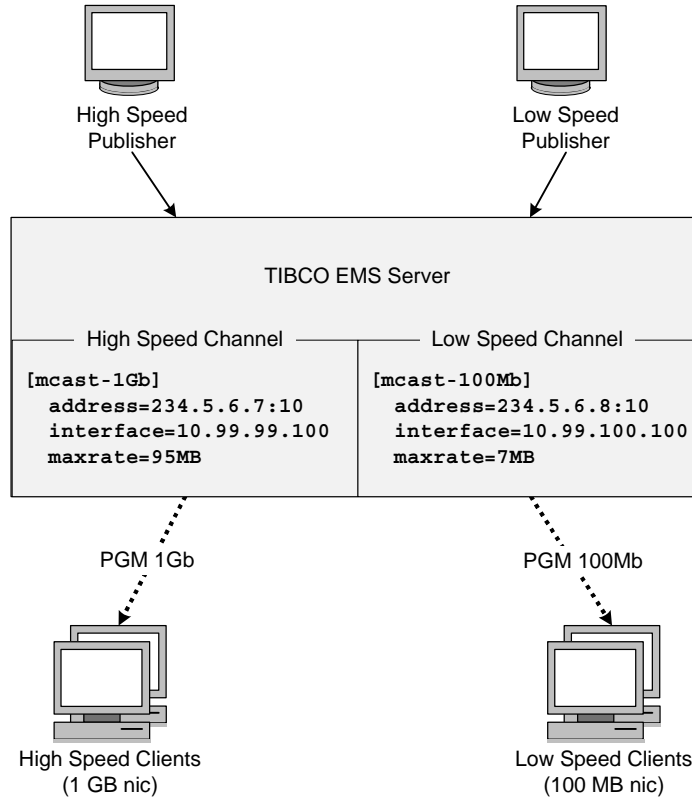
This example assumes that multicast connectivity exists and available bandwidth on the network is known. While not every aspect of a multicast deployment is covered in this example, it does illustrate the general thought process applied to multicast deployment.

Step 1: Design the Multicast Network Architecture

The location of the EMS server and clients are very important to a multicast deployment. You must ensure that multicast packets can get to all network nodes intended to receive multicast data, and you must account for all bandwidth across the network and network segments that the multicast data traverses. While TIBCO Enterprise Message Service detects and reports general connectivity problems, it is generally much easier to determine if there is connectivity before testing an EMS deployment. Your network administrator should be able to help with this.

For this example, let us assume that we are multicasting two streams of data: a fast data feed to some high performance processes on a 1 Gb network, and on a separate 100 Mb network a slower stream to a number of desktop applications. This leads us to the architecture shown in [Figure 20](#):

Figure 20 Sample Multicast Deployment Architecture



Note that two separate channels using different interfaces are to be configured at the server, allowing the server to simultaneously multicast on a high speed Gigabit network and a slower 100Mb network.

Step 2: Install and Set Up EMS

Installation is straightforward, as described in the *TIBCO Enterprise Message Service Installation*. The only requirement above a regular EMS installation is that the multicast daemon must be running on any machine that receives multicast data.

On Windows systems, you can register the multicast daemon as a service using the `emsntsrgr` utility. See [emsntsrgr](#) on page 112 for more information.

Setup the EMS Server

Before sending multicast data, first the EMS server needs to be configured. Configuring the EMS server requires you to change some global settings in the `tibemsd.conf` file, and to configure multicast channels in the `channels.conf` file. After channels are configured, you enable topics for multicast by setting their channel properties in the `topics.conf` file.

Enable the Server for Multicast

To begin, some general settings must be configured in the EMS server's main configuration file, `tibemsd.conf`:

- Enable multicast in the server by setting `multicast=enabled`.
- Enable multicast in the console trace by setting `console_trace=+MULTICAST`.

While enabling this trace is not required, it is very useful during the initial deployment, providing multicast-related warnings and errors.

- Enable flow control by setting `flow_control=enabled`.

Under heavy load, it is possible for publishers to feed data into the server faster than the server can multicast the data out. Enabling flow control causes the server to push back on the publishers, slowing them down if the server falls behind. This is not required, but highly suggested because it gives the server some room to minimize loss if this happens.

You should have added the following lines to the `tibemsd.conf`:

```
multicast=enabled
console_trace= DEFAULT,+MULTICAST
flow_control=enabled
```



You may also want to add `MULTICAST` to the server's `startup_abort_list`, if multicast is required in your architecture.

Configure Multicast Channels

The next step configures the multicast channels. In this example there are two multicast channels, `mcast-1Gb` and `mcast-100Mb`. The section [Sample channels.conf Settings](#) below shows specific settings for these steps:

1. Create the `channels.conf` file.

This file is described in [channels.conf on page 241](#).

2. Create two channels in the `channels.conf` file, `[mcast-1Gb]` and `[mcast-100Mb]`.
3. Set the address and destination port for each channel, using the `address` parameter.
4. Set the interface for each channel, using the `interface` parameter.

For this example, the server is on a multi-homed machine so we must explicitly specify interfaces for each channel. If an interface is not specified, the EMS server uses the default interface. Note that this is also true for the multicast daemon. Use the `-ifc` command line parameter when running multicast daemons on multi-homed machines, described in [Command Line Options on page 376](#).

5. Set a `maxrate` for each channel.

The `maxrate` parameter restricts the rate at which the server sends messages over the channel. See [Estimating the Maxrate](#) below for a discussion of how the maxrate was determined.

Sample channels.conf Settings

When you have completed your channel configuration, the `channels.conf` file should contain the following lines:

```
[mcast-1Gb]
  address=239.1.1.1:10
  interface=10.99.99.99
  maxrate=112MB

[mcast-100Mb]
  address=239.1.1.2:10
  interface=10.99.99.100
  maxrate=8MB
```

Estimating the Maxrate

In this example, we have set the `maxrate` properties using arbitrary network usage numbers to arrive at an estimate of network capacity. The process used to estimate the `maxrate` can be described as follows:

First find your average network usage, not including expected multicast data. This assumes metric data rate measurement.

- For the 1Gb network, let us assume about 10% usage, so 900Mb is available.
- On the 100Mb network, let us assume 30% usage, so 70Mb is available.

From here, calculate the available bytes per second for your network:

- $900 \text{ Mb} * 1\text{byte}/8\text{bits} \approx 112 \text{ MB (rounded down)}$
- $70 \text{ Mb} * 1\text{byte}/8\text{bits} \approx 8 \text{ MB (rounded down)}$

These initial rates are for testing purposes, and these will be modified later to maximize performance. Remember the cardinal rule with multicast performance is that sometimes you have to *slow down to speed up*. A rate that is too high will induce loss, which in turn causes messages to be resent, slowing the actual rate to something far below what your network is capable of.



This example uses only one channel per network. If your architecture has multiple multicast groups (channels with different address properties), remember to include all channels on the network in your maximum bandwidth calculations. This may require some balancing of data rates across channels.

Configure Multicast Topics

After the channels are defined, you must set the `channel` properties for topics so the server will send messages using multicast to multicast-enabled consumers subscribed to the topics. The channel property is set in the `topics.conf` configuration file.

In this example, we use two topics, `feed-1Gb` and `feed-100Mb`. These topic names are arbitrary; the key is assigning the correct channels to the topics.

Sample
topics.conf

```
feed-1Gb channel=mcast-1Gb
feed-100Mb channel=mcast-100Mb
```

EMS Client Setup

There are two main requirements for EMS clients to receive multicast data:

- The client must use a session mode of `NO_ACKNOWLEDGE` when subscribing to the multicast topic. See [Creating a Multicast Consumer on page 379](#) for more information.
- A multicast daemon must be running on the same computer as the client. See [Starting the Multicast Daemon on page 379](#).

TIBCO Software also highly suggests that applications take advantage of the multicast exception listener to be notified of multicast related events, errors, and warnings. This is accomplished in two simple steps, illustrated in java code below:

1. First, create a class that implements [TibjmsMulticastExceptionListener](#).

```
class MulticastExceptionHandler implements
com.tibco.tibjms.TibjmsMulticastExceptionListener
{
    public void onMulticastException(Connection connection,
                                     Session session,
                                     MessageConsumer consumer,
                                     JMSEException e)
    {
        System.out.println(e.getMessage());
    }
}
```

2. Next, set the multicast exception listener. Ideally, this will be done before you create a consumer of a multicast enabled topic.

```
com.tibco.tibjms.Tibjms.setMulticastExceptionListener(new
MulticastExceptionHandler());
```

To set up a multicast exception listener using the C API, see the *TIBCO Enterprise Message Service C & COBOL API Reference*.

To set up a multicast exception listener using the .NET API, see the *TIBCO Enterprise Message Service .NET API Reference*, available through the HTML documentation interface.

Step 3: Determine Network and Application Capabilities

It is valuable to know what EMS data rates the network can accommodate. If your application can handle data at least as fast as your network can, you will encounter the unusual situation where the network is your throughput bottleneck, which is ideal—as long as those data rates meet your requirements.

Determine Network Capabilities

Now that the server is enabled, you can test and fine tune the maxrate specified for the channels. This section describes one method for testing your settings.

This example assumes that the messages multicast on the network are small, on average 100 bytes per message.

These steps describe how to test the network bandwidth settings:

1. Start the EMS server using the `-trace FLOW` option, as described in [Starting the EMS Server Using Options on page 109](#).
2. From the command line, start the multicast daemon, using the `tibemsmcd -trace` command.

Using the `-trace` option is not required, but may assist in detecting any problems. See [Starting the Multicast Daemon on page 379](#) for more information.

3. On each node receiving multicast data, open a command line window and navigate to the `TIBCO_HOME/ems/8.0/samples/java` folder:
4. Launch the `tibjmsPerfSlave` sample program included with EMS:

```
> java tibjmsPerfSlave -server serverURL
```

It is very important to run the `jmsPerfSlave` application on every node that will receive multicast data. EMS Multicast must be tuned to perform at the level of the slowest receiver, or congestion and loss can occur.

5. On each node publishing multicast data, launch:

```
> java tibjmsPerfMaster -topic feed-1Gb -channel mcast-1Gb
-ackmode NO -time 30 -size 100
```

or

```
> java tibjmsPerfMaster -topic feed-100Mb -channel mcast-100Mb
-ackmode NO -time 30 -size 100
```

These performance applications should be run on each node the publisher will run.

6. Review the server and multicast daemon output for any warnings or errors. If you see any trace messages indicating loss, or if drastic rate fluctuations occur, this usually means you may be exceeding the maximum rate selected.

For example, a multicast error might look like:

```
channel='mcast-1Gb', Loss Detected, status=IO failed
```

On the server it is typical to see the following:

```
2008-11-13 17:11:57.300 Multicast channel 'mcast-100Mb' has
exceeded its allotted bandwidth
```

If flow control is and FLOW tracing are enabled, you should see the following as well:

```
2008-11-13 17:11:57.781 Flow control engaged on topic
'feed-100Mb'
....
```

When flow control is enabled, this simply means that the server is pushing back on the publisher to slow down to the rate defined by the multicast channel.

When the trace messages indicate that multicast channels have exceeded their bandwidth, this indicates that the channel `maxrate` is too low—your publisher is publishing faster than the channel's `maxrate` allows. On the other hand, when the `maxrate` is too high, you will see errors indicating that loss is detected.

Depending on what the trace messages show, try adjusting the maximum rate of the channel (the `maxrate` property) up or down, and repeat this test.

Evaluate Multicast Receiver Applications

One key to a successful multicast deployment is ensuring that the EMS server does not overrun your applications with data. This frequently this means setting the delivery rates (the channel's `maxrate` property) to a rate below what your network and EMS alone can handle.



The channel's maximum delivery rate, or `maxrate`, should not exceed the rate at which the slowest message consumer can consume incoming messages.

Determining the maximum message rate that your slowest application can handle reduces the time spent during trial and error testing. If your applications can process data faster than the network can deliver it, you will have already determined the maximum rate from determining your network capabilities.

Largely, determining the maximum speed at which the slowest application can process incoming data is a trial and error process. It is often useful to programmatically determine an application's maximum rate of consumption. The multicast daemon buffers messages for slower applications, but this increases the latency of data and memory usage of the multicast daemon, and is not considered a sustainable condition.

If a multicast-enabled consumer is expected to fall behind at times and can sustain loss, you can account for this using the `maxbytes` and `maxmsgs` properties for topics. See [Destination Properties on page 58](#) for details about these properties.

Tune Channel Parameters

Once you have determined network capabilities and multicast receiver rates, you can experiment with increasing (or sometimes decreasing) channel [maxrate](#) properties to achieve maximum throughput. Finding the maximum multicast rate your environment can handle often requires more experimentation than anything else. Always remember that once the network has been saturated, throughput will drastically drop.

Tuning the Operating System

Unfortunately, operating systems are not normally tuned for high performance with raw sockets. There are a number of performance changes you can make; typically, these changes involve socket buffering and can yield significant increases in throughput.

For example, on Linux one can modify window sizes in the `/etc/sysctl.conf` file:

```
net.core.wmem_max=1073741824
net.core.rmem_max=1073741824
net.core.wmem_default=1073741824
net.core.rmem_default=1073741824
```

However, operating system tuning for multicast falls outside of the scope of this document. The TIBCO Professional Services Group can provide assistance with advanced tuning specific for TIBCO Enterprise Message Service, and there are many resources on the internet for general tuning of operating systems concerning network performance.

Development and Production Environments

Configuring multicast is specific to a particular network, and your configuration must account for traffic patterns and characteristics of nodes that are unique to your network. Consequently, the tuning parameters applied to a development environment may not be optimum in a production environment, and the reverse is also true. When migrating from one environment to another, it is important to remember that although the application and EMS architecture pattern may be identical, the network and application capabilities will need to be reevaluated through the repetition of the steps described in this section. Topic and channel definition names should remain the same, but rate, interface, and timeout parameters for multicast must be reevaluated.

The channel properties that should be reevaluated upon deployment include:

- [maxrate on page 242](#)
- [ttl on page 242](#)
- [interface on page 243](#)

Troubleshooting EMS Multicast

Multicast deployment issues are often more difficult to resolve than similar unicast issues. Reasons for the additional difficulty include:

- Older networking equipment that was not designed with multicast deployment in mind. For example, switches that can only flood multicast or routers that do not have modern multicast routing protocols.
- Different equipment may solve the same problem in different ways. For example, some switches use IGMP snooping while others use CGMP.
- Multicast diagnostic tools are not readily available.
- Network administrators may not be as experienced in multicast deployment issues as they are with unicast deployment.
- Bandwidth is automatically shared equitably among competing unicast streams, but administrator intervention may be required to achieve desired multicast bandwidth sharing.

Troubleshooting Tips

This section give some troubleshooting tips to help you respond to difficulties you may experience with your multicast deployment.

General Tips

If you are experiencing problems with your deployment, begin with these practices:

- The "bottom-up" approach generally seems best. That is, get the lowest layers of the network stack working first.
- Begin with the EMS server and trace your way through each switch and router to all receivers. Try moving your receiving application to the same hub as the server (not a switch or a router), and confirm that you have multicast connectivity. Once that works, move on to more complicated multicast networks.

Connectivity

EMS will detect multicast connectivity issues; it may take up to 64 seconds to detect a connectivity problem. These suggestions can help resolve issues with connectivity:

- Verify that the network has good unicast connectivity between the sender and all receivers before tackling multicast connectivity problems.
- Verify that IP Multicast is supported and enabled on your routers or switches and all networks interfaces that are being used.
- Verify that address scoping at the router is not preventing multicast packets from being forwarded.
- Test your multicast application without enabling multicast in the EMS server to determine if a more general topic or application configuration issue is preventing message reception. For example, a consumer that is consuming on the wrong topic.
- Enable multicast and topic tracing in the server to ensure proper configuration, and to verify that messages are being multicast by the server.
- Enable multicast daemon trace messages to check for any configuration issues, warnings, or errors.
- Ensure that you are using the proper interface(s) in the server and the multicast daemon. On a multi-homed host, it is possible that the default interface cannot receive multicast data from the server.
- Ensure that the channel's `ttl` is large enough for data to cross all of your switches and routers.

Data Loss

These suggestions can help if you are experiencing data loss:

- Enable and check statistics to see if data is being delivered and whether excessive loss is encountered. If loss is detected, decreasing the multicast channel's `maxrate` property may alleviate the situation.
- Make sure that multicast streams are being generated with a time to live that is long enough for messages to reach their destination using the longest-possible path through the network.
- If you see increased loss as multicast rates go up, look for routers or switches that might be configured to limit the broadcast rate. These generally limit the multicast rate too. For example, Cisco Catalyst 5000 series switches can be configured to limit the packet per second or percentage of broadcast/multicast traffic with the `set port broadcast` command.

Application and Multicast Daemon Errors and Warnings

You may find these tips useful if you are experiencing errors in the multicast daemon or client application:

- Register a multicast exception listener in the receiving application. This provides the application with a way to detect, log, and handle multicast warnings and errors.

Note that multicast events are also logged at the client if client trace is enabled on the server, but that comes at a performance price and can cause other problems. For this reason, we do not recommend using client trace outside of debugging basic connectivity issues or as directed by TIBCO support.

- Typically, when consumer creation fails for a consumer on a multicast-enabled topic, a message is written to the multicast daemon's log (or console) as well as to the server log. An appropriate exception or return code is generated from the call on the client as well. After eliminating the other non-multicast related reasons (security, general configuration) you may want to check:

- Is the multicast daemon running?
- Is the multicast daemon running on the correct port?
- Did channel creation in the multicast daemon fail? (This indicates a protocol level multicast problem.)

- When the multicast daemon detects excessive loss, the multicast connection exception `IO Failed` is generated in the application. Usually, this means that the server is sending too fast, and `maxrate` for the channel needs to be decreased. The multicast daemon will report an error, similar to the following:

```
2007-10-02 16:45:09.551 Multicast error: channel='mcast', Loss
Detected, status=IO failed
```

You will also notice in the multicast statistics that the particular channel's `rcv_losses` are growing.

- If a consumer receives a multicast exception of `TIBEMS_TIMEOUT` with a message similar to `Timeout reached` which may indicate a configuration or hardware problem, this indicates a lack of multicast connectivity. While unicast connectivity exists between the client and server and the multicast channel was set up, multicast data cannot get from the server to the local multicast daemon. Note that this may take more than a minute to detect.
- Start a subscriber listening to `$sys.monitor.multicast.stats` monitoring messages to receive multicast-related statistics.

Server Errors

In General, server errors are self-descriptive. It is important to note that client errors may be returned to the server to be logged, providing a centralized place to look for multicast errors. However, these errors do not include minor loss on a particular client, or loss of messages from a client failover.

Chapter 15 **Working With TIBCO Rendezvous**

This chapter describes the interoperation of EMS and TIBCO Rendezvous.

Topics

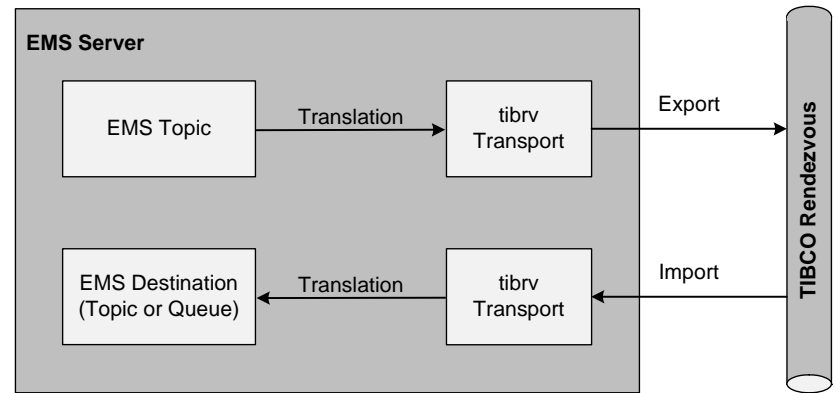
- [Overview, page 402](#)
- [Configuring Transports for Rendezvous, page 404](#)
- [Topics, page 410](#)
- [Queues, page 412](#)
- [Import Issues, page 414](#)
- [Export Issues, page 416](#)
- [Message Translation, page 417](#)
- [Pure Java Rendezvous Programs, page 423](#)

Overview

TIBCO Enterprise Message Service (release 4 and later) can exchange messages with TIBCO Rendezvous (release 6.9 and later).

- Scope
- EMS can import and export messages to an external system through an EMS *topic*.
 - EMS can import messages from an external system to an EMS *queue* (but queues cannot export).

Figure 21 Rendezvous Transports in the EMS Server



Message Translation

EMS and Rendezvous use different formats for messages and their data. When `tibemspd` imports or exports a messages, it translates the message and its data to the appropriate format; for details, see [Message Translation on page 438](#).

Configuration

`tibemspd` uses definitions and parameters in four configuration files to guide the exchange of messages with Rendezvous.

- Enabling
- The parameter `tibrv_transports` (in the configuration file `tibemspd.conf`) globally enables or disables message exchange with Rendezvous. The default value is `disabled`. To use these transports, you must explicitly set this parameter to `enabled`.

Transports	Transport definitions (in the configuration file transports.conf) specify the communication protocol between EMS and the external system; for details, see Configuring Transports for Rendezvous on page 404 .
Destinations	<p>Destination definitions (in the configuration files topics.conf and queues.conf) can set the <code>import</code> and <code>export</code> properties to specify one or more transports:</p> <ul style="list-style-type: none">• <code>import</code> instructs <code>tibemsd</code> to import messages that arrive on those transports from Rendezvous, and deliver them to the EMS destination.• <code>export</code> instructs <code>tibemsd</code> to take messages that arrive on the EMS destination, and export them to Rendezvous via those transports. <p>For details, see Topics on page 433, and Queues on page 434.</p>
RVCM Listeners	<p>When exporting messages on a transport configured for certified message delivery, you can pre-register RVCM listeners in the file <code>tibrvcn.conf</code>.</p> <p>For details, see tibrvcn.conf on page 257, and Certified Messages on page 416</p>

Configuring Transports for Rendezvous

Transports mediate the flow of messages between EMS and TIBCO Rendezvous.

`timemsd` connects to Rendezvous daemons in the same way as any other Rendezvous client would. Transport definitions (in the file `transports.conf`) configure the behavior of these connections. You must properly configure these transports.

How Rendezvous
Messages are
Imported

The EMS server connects to the Rendezvous daemon as any other Rendezvous client would. Messages received from the Rendezvous daemon are stored in Rendezvous queues, then are dispatched to callbacks. The EMS server creates JMS message copies of the Rendezvous messages, and begins processing them as EMS messages. Transports determine how messages are imported.

Rendezvous messages that are imported through a transport are held in queues specific to that transport. Each transports is associated with a different Rendezvous queue, which holds as many Rendezvous messages as necessary. The number of pending messages in the queue will grow if the rate of incoming Rendezvous messages is greater than the rate at which the EMS server is able to process the corresponding EMS messages.

Depending on the import delivery mode defined for the transport, the EMS messages will be persisted on disk, which increases the likelihood of backlog in the Rendezvous queues, and which in turn results in a EMS process memory growth. This memory growth is not accounted for in any of the EMS server statistics.

Queue Limit
Policies

In order to limit the number of pending messages in Rendezvous queues, a transport property allows you to set a queue limit policy, as you would for TIBCO Rendezvous client applications. When the queue limit for the transport is reached, the Rendezvous library discards a set number of messages. The default policy is `TIBRVQUEUE_DISCARD_NONE`, which means that no message is ever discarded. Setting `TIBRVQUEUE_DISCARD_FIRST` or `TIBRVQUEUE_DISCARD_LAST` allows you to specify the maximum number of Rendezvous messages that can be pending in the queue before the discard policy that you have selected is applied. When the limit is reached, the number of messages discarded is based on the discard amount value.

When the limit is reached, Rendezvous messages are discarded, and so are not imported as EMS messages, regardless of the EMS import delivery mode. As stated above, a Rendezvous message becomes a EMS message only after it has been dispatched from the Rendezvous queue. If a queue limit is exceeded, reliable Rendezvous messages are lost.

Rendezvous certified messages are not lost, but the message flow is interrupted. The redelivery of the missed messages is handled automatically by the Rendezvous libraries, and can not be controlled by the EMS server.

Reaching a queue limit also generates a Rendezvous advisory that is logged (see RVADV log and console trace in the TIBCO Rendezvous documentation), indicating which transport reached its queue limit. This advisory goes into an independent, non limited, Rendezvous queue. If lots of advisories are generated, this internal queue may also grow, signalling that the limit policy is not appropriate for your environment.

Take care when setting a queue limit policy. In a controlled environment where the risk of Rendezvous producers overwhelming the EMS server is low, there is no need to set a queue limit policy.

Transport Definitions

`transports.conf` contains zero or more transport definitions. Each definition begins with the name of a transport, surrounded by square brackets. Subsequent lines set the parameters of the transport.

Table 62 Rendezvous: Transport Parameters (Sheet 1 of 4)

Parameter	Description
type	Required. For Rendezvous transports, the value must be either <code>tibrv</code> or <code>tibrvcn</code> .
Rendezvous Parameters	
Use these properties for either <code>tibrv</code> or <code>tibrvcn</code> transports.	
The syntax and semantics of these parameters are identical to the corresponding parameters in Rendezvous clients. For full details, see the Rendezvous documentation set.	
service	When absent, the default value is 7500.
network	When absent, the default value is the host computer's primary network.

Table 62 Rendezvous: Transport Parameters (Sheet 2 of 4)

Parameter	Description
daemon	<p>When absent, the default value is an <code>rvd</code> process on the local host computer. When transporting messages between EMS and Rendezvous, the <code>rvd</code> process must be configured to run on the same host as the EMS daemon (<code>tibemsd</code>).</p> <p>To connect to a non-default daemon, supply <code>hostname:protocol:port</code>. You may omit any of the three parts. The default <code>hostname</code> is the local host computer. The default protocol is <code>tcp</code>. The default <code>port</code> is 7500.</p>

Rendezvous Certified Messaging (RVCN) Parameters

Use these properties only for `tibrvcn` transports.

The syntax and semantics of these parameters are identical to the corresponding parameters in Rendezvous CM clients. For full details, see the Rendezvous documentation set.

cm_name	The name of the correspondent RVCN listener transport.
rv_tport	Required. Each RVCN transport depends in turn upon an ordinary Rendezvous transport. Set this parameter to the name of a Rendezvous transport (type <code>tibrv</code>) defined in the EMS configuration file <code>transports.conf</code> .
ledger_file	Name for file-based ledger.
sync_ledger	<code>true</code> or <code>false</code> . If <code>true</code> , operations that update the ledger do not return until changes are written to the storage medium.
request_old	<code>true</code> or <code>false</code> . If <code>true</code> , this transport server requests unacknowledged messages sent from other RVCN senders while this transport was unavailable.
default_ttl	This parameter sets default CM time limit (in seconds) for all CM messages exported on this transport.
explicit_config_only	<p><code>true</code> or <code>false</code>. If <code>true</code>, <code>tibemsd</code> allows RVCN listeners to register for certified delivery only if they are configured in advance with the EMS server (either in <code>tibrvcn.conf</code> or using the <code>create rvcnlistener</code> command). That is, <code>tibemsd</code> ignores registration requests from non-configured listeners.</p> <p>If <code>false</code> (the default), <code>tibemsd</code> allows any RVCN listener to register.</p>

Table 62 Rendezvous: Transport Parameters (Sheet 3 of 4)

Parameter	Description
EMS Parameters	
Use these properties for either <code>tibrv</code> or <code>tibrvcm</code> transports.	
<code>topic_import_dm</code> <code>queue_import_dm</code>	<p>EMS sending clients can set the <code>JMSDeliveryMode</code> header field for each message. However, Rendezvous clients cannot set this header. Instead, these two parameters determine the delivery modes for all topic messages and queue messages that <code>tibemsd</code> imports on this transport.</p> <p><code>TIBEMS_PERSISTENT</code> <code>TIBEMS_NON_PERSISTENT</code> <code>TIBEMS_RELIABLE</code></p> <p>When absent, the default is <code>TIBEMS_NON_PERSISTENT</code>.</p>
<code>export_headers</code>	<p>When <code>true</code>, <code>tibemsd</code> includes JMS header fields in exported messages.</p> <p>When <code>false</code>, <code>tibemsd</code> suppresses JMS header fields in exported messages.</p> <p>When absent, the default value is <code>true</code>.</p>
<code>export_properties</code>	<p>When <code>true</code>, <code>tibemsd</code> includes JMS properties in exported messages.</p> <p>When <code>false</code>, <code>tibemsd</code> suppresses JMS properties in exported messages.</p> <p>When absent, the default value is <code>true</code>.</p>

Table 62 Rendezvous: Transport Parameters (Sheet 4 of 4)

Parameter	Description
rv_queue_policy	<p>Set the queue limit policy for the Rendezvous queue used by the transport to hold incoming Rendezvous messages. This parameter has three parts:</p> <p><i>policy: max_msgs: qty_discard</i></p> <p>where <i>policy</i> is one of the queue limit policies described below, <i>max_msgs</i> is the maximum number of messages permitted in the queue before discard, and <i>qty_discard</i> is the number of messages that the EMS server discards when <i>max_msgs</i> is reached.</p> <p>The queue limit policies are:</p> <ul style="list-style-type: none">• TIBRVQUEUE_DISCARD_NONE — do not discard messages. Use this policy when the queue has no limit on the number of messages it can contain.• TIBRVQUEUE_DISCARD_FIRST — discard the first message in the queue. The first message in the queue is the oldest message, which if not discarded would be the next message dispatched from the queue.• TIBRVQUEUE_DISCARD_LAST — discard the last message in the queue. The last message is the most recent message received into the queue. <p>For example, the following would cause the Rendezvous library to discard the 100 oldest messages in the queue when the total number of messages in the queue reached 10,000:</p> <p><code>rv_queue_policy=TIBRVQUEUE_DISCARD_FIRST:10000:100</code></p> <p>If the <code>rv_queue_policy</code> is not present, the default queue limit policy is <code>TIBRVQUEUE_DISCARD_NONE</code>.</p>
temp_destination_timeout	<p>Specifies the amount of time the server is to keep the temporary destination (created for the RV inbox) after its last use of the destination. This is useful for a multi-server configuration. For example, in a configuration in which <code>rv-requester -> serverA -> serverB -> rv-responder</code>, setting <code>temp_destination_timeout=60</code> on <code>serverB</code> specifies that <code>serverB</code> is to hold the temporary destination for 60 seconds.</p>

Example

These examples from `transports.conf` illustrate the syntax of transport definitions.

```
[RV01]
type = tibrv
topic_import_dm = TIBEMS_RELIABLE
queue_import_dm = TIBEMS_PERSISTENT
service = 7780
network = lan0
daemon = tcp:host5:7885
```

```
[RV02]
type = tibrv
service = 7890
network = lan0
daemon = tcp:host5:7995
temp_destination_timeout = 60
```

```
[RVC01]
type = tibrvc
export_headers = true
export_properties = true
rv_tport = RV02
cm_name = RVCMTans1
ledger_file = ledgerFile.store
sync_ledger = true
request_old = true
default_ttl = 600
```

In the following two examples, RVC03 is an RVC transport which does not define a queue limit policy, but references the RV transport RV03, which *does* have a queue limit policy. If Rendezvous messages are published to a subject that in EMS has the destination property `import=RVC03`, no Rendezvous message will ever be discarded because each transport uses its own queue. Only messages that are imported directly through the RV03 transport will potentially be discarded, should the queue limit of 10000 messages be reached.

```
[RV03]
type = tibrv
service = 7890
network = lan0
daemon = tcp:host5:7995
rv_queue_policy = TIBRVQUEUE_DISCARD_LAST:10000:100
```

```
[RVC03]
type = tibrvc
rv_tport = RV03
cm_name = RVCMTans2
ledger_file = ledgerFile2.store
sync_ledger = true
request_old = true
default_ttl = 600
```

Topics

Topics can both export and import messages. Accordingly, you can configure topic definitions (in the configuration file `topics.conf`) with `import` and `export` properties that specify one or more external transports:

- `import`
 - `import` instructs `tibemsd` to import messages that arrive on those transports from Rendezvous, and deliver them to the EMS destination.
- `export`
 - `export` instructs `tibemsd` to take messages that arrive on the EMS destination, and export them to Rendezvous via those transports.



The EMS server *never* re-exports an imported message on the same topic.

(For general information about `topics.conf` syntax and semantics, see [topics.conf on page 257](#). You can also configure topics using the administration tool command `addprop topic`.)

Example For example, the following `tibemsd` commands configure the topic `myTopics.news` to import messages on the transports `RV01` and `RV02`, and to export messages on the transport `RV02`.

```
addprop topic myTopics.news import="RV01,RV02"
addprop topic myTopics.news export="RV02"
```

Rendezvous messages with subject `myTopics.news` arrive at `tibemsd` over the transports `RV01` and `RV02`. EMS clients can receive those messages by subscribing to `myTopics.news`.

EMS messages sent to `myTopics.news` are exported to Rendezvous over transport `RV02`. Rendezvous clients of the corresponding daemons can receive those messages by subscribing to `myTopics.news`.

Import Only when Subscribers Exist

When a topic specifies `import` on a connected transport, `tibemsd` imports messages only when the topic has registered subscribers.

Wildcards

Wildcards in the `import` and `export` properties obey EMS syntax and semantics (which is identical to Rendezvous syntax and semantics); see [Destination Name—Syntax and Semantics on page 432](#).

Certified Messages

You can [import](#) and [export](#) TIBCO Rendezvous certified messages (`tibrvc` transport) to EMS topics. Rendezvous certified transports guarantee message delivery.

RVCM Ledger `tibrvc` transports can store information about subjects in a ledger file. You can review the ledger file using an administration tool command; see [show `rvc`transportledger](#) on page 167).

For more information about ledger files, see TIBCO Rendezvous documentation.

Subject Collisions Subscribers to destinations that import from RVCM transports are subject to the same restrictions that direct RVCM listeners. These restrictions are described in the TIBCO Rendezvous documentation, and include subject collisions.

When importing messages from RV, the EMS server creates RVCM listeners using a single name for each transport. This can result in subject collisions if the corresponding EMS subscribers have overlapping topics.

Queues

Queues can **import** messages, but cannot **export** them.

Configuration

You can configure queue definitions (in the configuration file `queues.conf`) with the `import` property that specify one or more external transports.

- **import** instructs `tibemsd` to import messages that arrive on those transports from Rendezvous, and deliver them to the EMS destination.

(For general information about `queues.conf` syntax and semantics, see [queues.conf on page 250](#). You can also configure queues using the administration tool command `addprop queue`.)

Example For example, the following `tibemsdadmin` command configures the queue `myQueue.in` to import messages on the transports `RV01` and `RV02`.

```
addprop queue myQueue.in import="RV01,RV02"
```

Rendezvous messages with subject `myQueue.in` arrive at `tibemsd` over the transports `RV01` and `RV02`. EMS clients can receive those messages by subscribing to `myQueue.in`.

Import—Start and Stop

When a queue specifies **import** on a connected transport, `tibemsd` immediately begins importing messages to the queue, even when no receivers exist for the queue.

For static queues (configured by an administrator) `tibemsd` continues importing until you explicitly delete the queue.

Wildcards

Wildcards in the **import** property obey EMS syntax and semantics (not Rendezvous syntax and semantics); see [Destination Name—Syntax and Semantics on page 432](#).

EMS clients cannot subscribe to wildcard queues—however, you can define wildcard queues in the EMS server for the purpose of property inheritance. That is, you can configure a static queue named `foo.*` and set properties on it, so that child queues named `foo.bar` and `foo.baz` will both inherit those properties.

If you define a queue that imports `foo.*`, `tibemsd` begins importing all matching messages from Rendezvous. As messages arrive, `tibemsd` creates dynamic child queues (for example, `foo.bar` and `foo.baz`) and delivers the messages to them. Notices that `tibemsd` delivers messages to these dynamic child queues even when no consumers exist to drain them.

Import Issues

This section presents issues associated with importing messages to EMS from Rendezvous—whether on a topic or a queue.

Field Identifiers

When importing and translating Rendezvous messages, `tibemsd` is only able to process standard message field types that are identified by name in the Rendezvous program application. Custom fields and fields identified using a field identifier cannot be imported to EMS.

JMSDestination

When `tibemsd` imports and translates a Rendezvous message, it sets the `JMSDestination` field of the EMS message to the value of the Rendezvous subject. Therefore, imported destination names must be unique. When a topic and a queue share the same name, *at most one* of them may set the `import` property. For example, if a topic `foo.bar` and a queue `foo.bar` are both defined, only one may specify the `import` property.

JMSReplyTo

When `tibemsd` imports and translates a Rendezvous message, it sets the `JMSReplyTo` field of the EMS message to the value of the Rendezvous reply subject, so that EMS clients can reply to the message.

Usually this value represents a Rendezvous subject. You must explicitly configure `tibemsd` to create a topic with a corresponding name, which exports messages to Rendezvous.

JMSExpiration

When `tibemsd` imports and translates a Rendezvous certified message, it sets the `JMSExpiration` field of the EMS message to the time limit of the certified message.

If the message time limited is exceeded, the sender program no longer certifies delivery.

Note that if the `expiration` property is set for a destination, it will override the `JMSExpiration` value set by the message producer.

JMSTimestamp

When `tibemsd` imports and translates a Rendezvous message, it uses the `JMSTimestamp` header field to determine when the message was created. If the `JMSTimestamp` field is not set, the `tibemsd` ignores the expiration field, because expiration is based on an unknown creation time.

The Rendezvous sender must create a field called `JMSTimestamp` in order to enable message expiration.

Guaranteed Delivery



For full end-to-end certified delivery from Rendezvous to EMS, all three of these conditions must be true:

- Rendezvous senders must send labeled messages on RVCM transports. See the *TIBCO Rendezvous Concepts* manual for more information.
- The transport definition must set `topic_import_dm` or `queue_import_dm` (as appropriate) to `TIBEMS_PERSISTENT`.
- Either a durable queue or a subscriber for the EMS topic must exist.

Export Issues

This section presents issues associated with exporting messages from EMS to Rendezvous.

JMSReplyTo

Topics	Consider an EMS message in which the field <code>JMSReplyTo</code> contains a topic. When exporting such a message to Rendezvous, you must explicitly configure <code>tibemsd</code> to import replies from Rendezvous to that reply topic.
Temporary Topics	Consider an EMS message in which the field <code>JMSReplyTo</code> contains a temporary topic. When <code>tibemsd</code> exports such a message to Rendezvous, it <i>automatically</i> arranges to import replies to that temporary topic from Rendezvous; you do not need to configure it explicitly.

Certified Messages

RVC Registration	<p>When an RVC listener receives its first labeled message, it registers to receive subsequent messages as certified messages. Until the registration is complete, it receives labeled messages as reliable messages. When exporting messages on a <code>tibrvc</code> transport, we recommend either of two actions to ensure certified delivery for all exported messages:</p> <ul style="list-style-type: none"> • Create the RVC listener before sending any messages from EMS clients. • Pre-register an RVC listener, either with the administration tool (see create rvcmlistener on page 132), or in the configuration file <code>tibrvc.conf</code> (see tibrvc.conf on page 257).
---------------------	---

Guaranteed Delivery



For full end-to-end certified delivery to Rendezvous from EMS, the following condition must be true:

- EMS senders must send persistent messages.

Message Translation

JMS Header Fields

EMS supports the 11 predefined JMS header fields; see [JMS Message Header Fields on page 17](#).

Special Cases	<p>These header fields are special cases:</p> <ul style="list-style-type: none"> JMS header <code>JMSDestination</code> corresponds to Rendezvous subject. JMS header <code>JMSReplyTo</code> corresponds to Rendezvous reply subject. JMS header <code>JMSExpiration</code> corresponds to the time limit of the Rendezvous certified message.
Import	When importing a Rendezvous message to an EMS message, <code>tibemsd</code> does not set any JMS header fields, except for the special cases noted above.
Export	<p>When exporting an EMS message to a Rendezvous message, <code>tibemsd</code> groups all the JMS header fields (except for the special cases noted above) into a single submessage within the Rendezvous message. The field <code>JMSHeaders</code> contains that submessage. Fields of the submessage map the names of JMS header fields to their values.</p> <p><code>tibemsd</code> ignores any JMS header fields that are null or absent—it omits them from the exported message.</p> <p>You can instruct <code>tibemsd</code> to suppress the entire header submessage in all exported messages by setting the transport property <code>export_headers = false</code>.</p> <p>Table 63 presents the mapping of JMS header fields to Rendezvous data types (that is, the type of the corresponding field in the exported message).</p>

Table 63 Rendezvous: Mapping JMS Header Fields to RV Datatypes (Sheet 1 of 2)

JMS Header Name	Rendezvous Type
<code>JMSDeliveryMode</code>	<code>TIBRVMSG_U8</code>
<code>JMSDeliveryTime</code>	<code>TIBRVMSG_U64</code>
<code>JMSPriority</code>	<code>TIBRVMSG_U8</code>
<code>JMSTimestamp</code>	<code>TIBRVMSG_U64</code>
<code>JMSExpiration</code>	<code>TIBRVMSG_U64</code>
<code>JMSType</code>	<code>TIBRVMSG_STRING</code>

Table 63 Rendezvous: Mapping JMS Header Fields to RV Datatypes (Sheet 2 of 2)

JMS Header Name	Rendezvous Type
JMSMessageID	TIBRVMSG_STRING
JMSCorrelationID	TIBRVMSG_STRING
JMSRedelivered	TIBRVMSG_BOOL
JMSDestination	send subject in TIBCO Rendezvous
JMSReplyTo	reply subject in TIBCO Rendezvous

JMS Property Fields

- Import
- When importing a Rendezvous message to an EMS message, `tibemsd` sets these JMS properties:
- `JMS_TIBCO_IMPORTED` gets the value `true`, to indicate that the message did not originate from an EMS client.
 - `JMS_TIBCO_MSG_EXT` gets the value `true`, to indicate that the message *might* contain submessage fields or array fields.

Import RVCM

In addition to the two fields described above, when `tibemsd` imports a certified message on a `tibrvcn` transport, it can also set these properties (if the corresponding information is set in the Rendezvous message):

Table 64 Rendezvous Mapping Message Properties

Property	Description
JMS_TIBCO_CM_PUBLISHER	A string value indicating the correspondent name of the TIBCO Rendezvous CM transport that sent the message (that is, the sender name).
JMS_TIBCO_CM_SEQUENCE	A long value indicating the CM sequence number of an RVCN message imported from TIBCO Rendezvous.

Export

When exporting an EMS message to a Rendezvous message, `tibemsd` groups all the JMS property fields into a single submessage within the Rendezvous message. The field `JMSProperties` contains that submessage. Fields of the submessage map the names of JMS property fields to their values.

The `tibemsd` daemon ignores any JMS property fields that are not set, or are set to null—it omits them from the exported message.

You can instruct `tibemsd` to suppress the entire properties submessage in the exported message by setting the transport property `export_properties = false`.

Message Body

`tibemsd` can export messages with any JMS message body type to TIBCO Rendezvous. Conversely, `tibemsd` can import messages with any message type from TIBCO Rendezvous.

For information about JMS body types, see [JMS Message Bodies on page 22](#).

For information about the structure of messages, see [JMS Message Structure on page 17](#).

Import When importing a Rendezvous message, `tibemsd` translates it to an EMS message body type based on the presence of the field in [Table 65](#).

Table 65 Rendezvous: Mapping Message Types (Import)

Rendezvous Field	EMS Body Type
JMSBytes	JMSBytesMessage
JMSObject	JMSObjectMessage
JMSStream	JMSStreamMessage
JMSText	JMSTextMessage
None of these fields are present.	JMSMapMessage



The field names `DATA` and `_data_` are reserved. We strongly discourage you from using these field names in either EMS and Rendezvous applications, and especially when these two message transport mechanisms interoperate.



Only standard Rendezvous fields identified by name can be imported into EMS. Custom fields and fields identified in the Rendezvous application by field identifiers cannot be imported.

Export When exporting an EMS message, `tibemsd` translates it to a Rendezvous message with the following structure:

- The field `JMSHeaders` contains a submessage; see [JMS Header Fields on page 417](#). When the transport parameter `export_headers` is `false`, this field is omitted.

- The field `JMSProperties` contains a submessage; see [JMS Property Fields on page 418](#). When the transport parameter `export_properties` is false, this field is omitted.
- When translating the data fields of an EMS message, the results depend on the JMS body type. [Table 66](#) specifies the mapping.

Table 66 *Rendezvous: Mapping Message Types (Export)*

JMS Body Type	Export Translation
<code>BytesMessage</code>	<p>The message data translates to a byte array that contains the bytes of the original EMS message.</p> <p>The field <code>JMSBytes</code> receives this data. It has type <code>TIBRVMSG_OPAQUE</code>.</p>
<code>ObjectMessage</code>	<p>The message data translates to a byte array containing the serialized Java object.</p> <p>The field <code>JMSObject</code> receives this data. It has type <code>TIBRVMSG_OPAQUE</code>.</p>
<code>StreamMessage</code>	<p>The message data translates to a byte array that encodes the objects in the original EMS message.</p> <p>The field <code>JMSStream</code> receives this data. It has type <code>TIBRVMSG_OPAQUE</code>.</p>
<code>TextMessage</code>	<p>The message data translates to a UTF-8 string corresponding to the text of the original EMS message.</p> <p>The field <code>JMSText</code> receives this data. It has type <code>TIBRVMSG_STRING</code>.</p>
<code>MapMessage</code>	<p>The message data fields map directly to top-level fields in the Rendezvous message. The fields retain the same names as in the original EMS message.</p> <p>See also, EMS Extensions to JMS Messages on page 16.</p>

Data Types

[Table 67](#) presents the mapping between EMS datatypes and Rendezvous datatypes. The mapping is bidirectional, except for the Rendezvous types that have no corresponding EMS type (for these types the mapping is marked as unidirectional in the middle column of [Table 67](#)).

Table 67 Rendezvous: Mapping Data Types (Sheet 1 of 2)

EMS	Map	Rendezvous
Boolean		TIBRVMSG_BOOL
Byte		TIBRVMSG_I8
Short	<—	TIBRVMSG_U8
Short		TIBRVMSG_I16
Integer	<—	TIBRVMSG_U16
Integer		TIBRVMSG_I32
Long	<—	TIBRVMSG_U32
Long		TIBRVMSG_I64
Long	<—	TIBRVMSG_U64
Float		TIBRVMSG_F32
Double		TIBRVMSG_F64
Short	<—	TIBRVMSG_IPPORT16
Integer	<—	TIBRVMSG_IPADDR32
MapMessage		TIBRVMSG_MSG
Long	<—	TIBRVMSG_DATETIME
byte[]		TIBRVMSG_OPAQUE
java.lang.String		TIBRVMSG_STRING
byte[]	<—	TIBRVMSG_XML
byte[]	<—	TIBRVMSG_I8ARRAY

Table 67 Rendezvous: Mapping Data Types (Sheet 2 of 2)

EMS	Map	Rendezvous
short[]	<—	TIBRVMSG_U8ARRAY
short[]		TIBRVMSG_I16ARRAY
int[]	<—	TIBRVMSG_U16ARRAY
int[]		TIBRVMSG_I32ARRAY
long[]	<—	TIBRVMSG_U32ARRAY
long[]		TIBRVMSG_I64ARRAY
long[]	<—	TIBRVMSG_U64ARRAY
float[]		TIBRVMSG_F32ARRAY
double[]		TIBRVMSG_F64ARRAY

Pure Java Rendezvous Programs

TIBCO Enterprise Message Service is shipped with the `tibrvjms.jar` file that you can include in your TIBCO Rendezvous applications. This JAR file includes the implementation of the `com.tibco.tibrv.TibrvJMSTransport` class. This class extends the `com.tibco.tibrv.TibrvNetTransport` class and allows your pure Java Rendezvous programs to communicate directly with the EMS server instead of through `rva`.

the application must include `tibrvjms.jar` and EITHER `tibrvjweb.jar` OR `tibrvj.jar`, but CANNOT include `tibrvnative.jar`

To use the `TibrvJMSTransport` class, your application must include `tibrvjms.jar` (included with EMS) and either `tibrvjweb.jar` or `tibrv.jar` (included with TIBCO Rendezvous). Your application *cannot* include `tibrvnative.jar`.



You can use `TibrvJMSTransport` only in Rendezvous applications. This class is not intended for use in your EMS Java clients.

Both TIBCO Rendezvous and EMS must be purchased, installed, and configured before creating pure Java Rendezvous applications that use the `TibrvJMSTransport` class.

The `TibrvJMSTransport` class provides Rendezvous reliable communication only. Other types of communication, such as certified messaging, are not supported by this transport.

Applications using this transport can send messages to a topic on an EMS server that has the same topic name as the subject of the message. EMS topics receiving Rendezvous messages sent by way of the `TibrvJMSTransport` do not need to specify the `import` property. This transport cannot be used to send messages to JMS queues.

For more information about `TibrvNetTransport` and how to create use transports in TIBCO Rendezvous Java programs, see TIBCO Rendezvous documentation. For more information about the additional methods of [TibrvJMSTransport](#), see the *TIBCO Enterprise Message Service Java API Reference*.

Chapter 16 **Working With TIBCO SmartSockets**

This chapter describes the interoperation of TIBCO Enterprise Message Service and TIBCO SmartSockets.

Topics

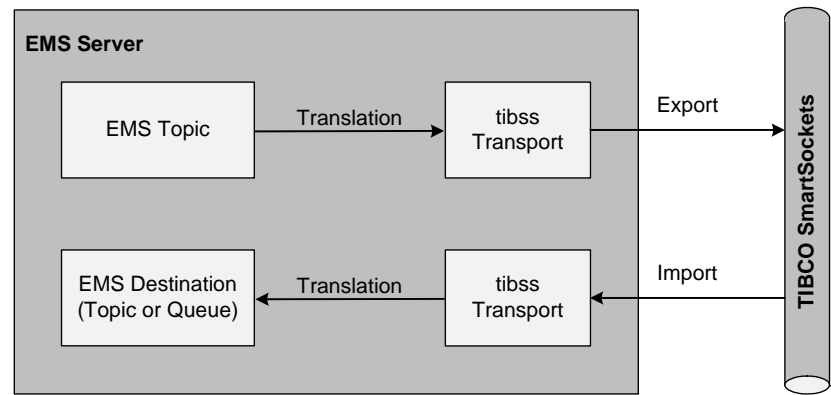
- [Overview, page 426](#)
- [Configuring Transports for SmartSockets, page 427](#)
- [Topics, page 433](#)
- [Queues, page 434](#)
- [Import Issues, page 436](#)
- [Export Issues, page 437](#)
- [Message Translation, page 438](#)

Overview

TIBCO Enterprise Message Service can exchange messages with TIBCO SmartSockets.

- Scope
- EMS can import and export messages to an external system through an EMS *topic*.
 - EMS can import messages from an external system to an EMS *queue* (but queues cannot export).

Figure 22 SmartSockets Transports in the EMS Server



Message Translation

EMS and SmartSockets use different formats for messages and their data. When `tibemsd` imports or exports a messages, it translates the message and its data to the appropriate format; for details, see [Message Translation on page 438](#).

Configuration

`tibemsd` uses definitions and parameters in three configuration files to guide the exchange of messages with SmartSockets.

- Enabling
- The parameter `tibss_transports` (in the configuration file `tibemsd.conf`) globally enables or disables message exchange with SmartSockets. The default value is `disabled`. To use these transports, you must explicitly set this parameter to `enabled`.

The parameter `tibss_config_dir` (in the configuration file `tibemsd.conf`) specifies the location of SmartSockets files needed by the SmartSockets client within `tibemsd`.

- | | |
|--------------|--|
| Transports | Transport definitions (in the configuration file <code>transports.conf</code>) specify the communication protocol between EMS and the external system; for details, see Configuring Transports for SmartSockets on page 427 . |
| Destinations | <p>Destination definitions (in the configuration files <code>topics.conf</code> and <code>queues.conf</code>) can set the <code>import</code> and <code>export</code> properties to specify one or more transports:</p> <ul style="list-style-type: none"> • <code>import</code> instructs <code>tibemsd</code> to import messages that arrive on those transports from SmartSockets, and deliver them to the EMS destination. • <code>export</code> instructs <code>tibemsd</code> to take messages that arrive on the EMS destination, and export them to SmartSockets via those transports. |

For details, see [Topics on page 433](#), and [Queues on page 434](#).

Starting the Servers

We recommend starting the SmartSockets RTserver before starting `tibemsd`.

Configuring Transports for SmartSockets

Transports mediate the flow of messages between TIBCO Enterprise Message Service and TIBCO SmartSockets.

`tibemsd` connects to SmartSockets RTservers in the same way as any other SmartSockets client. Transport definitions (in the file `transports.conf`) configure the behavior of these connections. You must properly configure these transports.

Transport Definitions

`transports.conf` contains zero or more transport definitions. Each definition begins with the name of a transport, surrounded by square brackets. Subsequent lines set the parameters of the transport.

Table 68 SmartSockets: Transport Parameters (Sheet 1 of 4)

Parameter	Description
type	Required. For SmartSockets transports, the value must be <code>tibss</code> .
SmartSockets Parameters	
The syntax and semantics of these parameters are identical to the corresponding parameters in SmartSockets clients. For full details, see the SmartSockets documentation set.	
server_names	<p>The value is a comma-separated list specifying connections to one or more SmartSockets RTservers.</p> <p>Each item in the list has the form <i>protocol:hostname:port</i>. You may omit any of the three parts. The default <i>hostname</i> is the local host computer. The default protocols and ports vary with hardware and operating system platforms; on Windows platforms, the default protocol is <code>tcp</code> and the default <i>port</i> is 5101.</p> <p>A list of several servers specifies fault tolerance—<code>timemsd</code> attempts to connect to them in the order listed.</p> <p>When this parameter is absent, the default instructs the EMS server to attempt to connect to an RTserver on the local host computer (the same computer as the EMS server), using default protocols and ports.</p>
username password	<code>timemsd</code> uses these two parameters to authenticate itself to the SmartSockets servers.
project	<p>SmartSockets uses projects to maintain orthogonal subject name-spaces.</p> <p>When absent, the default project is <code>rtworks</code>.</p>
delivery_mode	<p>This parameter determines the quality of service with which delivers messages to the SmartSockets server over this transport:</p> <p><code>best_effort</code> <code>gmd_all</code> <code>gmd_some</code> <code>ordered</code></p> <p>When absent, the default is <code>best_effort</code>.</p>

Table 68 SmartSockets: Transport Parameters (Sheet 2 of 4)

Parameter	Description
lb_mode	<p>SmartSockets servers balance the message load by distributing messages among several clients. This parameter determines the load balancing regimen for messages that this transport exports to the SmartSockets server.</p> <p>none round_robin weighted sorted</p> <p>When absent, the default is none.</p>
override_lb_mode	<p>enable instructs the RTserver to deliver all messages on this client connection—even if other clients participate in load balancing. For example, even though many order-processing clients might share the load of order messages, a message logging facility would require all order messages, rather than a subset.</p> <p>disable informs the RTserver that this client (that is, the EMS server) participates in load balancing (for example, sharing the load with other EMS servers).</p> <p>When absent, the default is enable.</p>
gmd_file_delete	<p>SmartSockets clients keep data for guaranteed message delivery (GMD) in a store file.</p> <p>disable instructs tibemsd to open the existing GMD store file.</p> <p>enable instructs tibemsd to delete the GMD store file and create a new one when creating this transport.</p> <p>When absent, the default is disable.</p>
import_ss_headers	<p>This parameter governs the import of SmartSockets message headers to EMS properties.</p> <p>The value can be none, type_num, or all. For complete details, see SmartSockets Message Properties on page 439.</p> <p>When absent, the default value is none.</p>

Table 68 SmartSockets: Transport Parameters (Sheet 3 of 4)

Parameter	Description
preserve_gmd	<p>This parameter determines the behavior of the EMS server when it has exported a GMD message to SmartSockets, and SmartSockets cannot deliver that message. When SmartSockets returns the undelivered message, EMS can either preserve it in the EMS undelivered message queue, or discard it.</p> <ul style="list-style-type: none">• <code>always</code> instructs EMS to preserve all undelivered GMD messages in the EMS undelivered message queue.• <code>receivers</code> instructs EMS to preserve only those undelivered GMD messages that SmartSockets could not deliver despite the existence of one or more GMD receivers. That is, if SmartSockets cannot deliver a message because no GMD receivers exist, then EMS does not preserve the undelivered message.• <code>never</code> instructs EMS to discard all undelivered SmartSockets GMD messages. <p>When absent, the default value is <code>never</code>.</p> <p>This parameter applies only when the transport's <code>delivery_mode</code> parameter is either <code>gmd_all</code> or <code>gmd_some</code>.</p> <p>When the EMS server preserves a GMD message, it follows these rules to convert the returned SmartSockets message to an EMS message:</p> <ul style="list-style-type: none">• Follow all general rules for importing messages; see Message Translation on page 438.• Disregard the value of the <code>import_ss_headers</code> parameter, and instead import all SmartSockets headers (as if the value of <code>import_ss_headers</code> were <code>all</code>). For a list of headers, see SmartSockets Message Properties on page 439.• Set the value of <code>JMS_TIBCO_SS_EXPIRATION</code> to the current time—that is, the time at which the SmartSockets server returned the undelivered message to EMS. (Notice that the this header would otherwise remain unused, since GMD messages do not expire.)

Table 68 SmartSockets: Transport Parameters (Sheet 4 of 4)

Parameter	Description
EMS Parameters	
topic_import_dm queue_import_dm	EMS sending clients can set the JMSDeliveryMode header field for each message. However, SmartSockets clients cannot set this header. Instead, these two parameters determine the delivery modes for all topic messages and queue messages that tibemsd imports on this transport. TIBEMS_PERSISTENT TIBEMS_NON_PERSISTENT TIBEMS_RELIABLE When absent, the default is TIBEMS_NON_PERSISTENT.
export_headers	When true, tibemsd includes JMS header fields in exported messages. When false, tibemsd suppresses JMS header fields in exported messages. When absent, the default value is true.
export_properties	When true, tibemsd includes JMS properties in exported messages. When false, tibemsd suppresses JMS properties in exported messages. When absent, the default value is true.

Example

These examples from `transports.conf` illustrate the syntax of transport definitions.

```
[SS01]
  type = tibss
  server_names = rtHost1
  username = emsServer6
  password = myPasswd
  project = sales_order_entry

[SS02]
  type = tibss
  server_names = tcp:rtHost2A:5555, ssl:rtHost2B:5571
  username = emsServer6
  password = myPasswd
  project = mfg_process_control
  override_lb_mode = enable
  delivery_mode = gmd_some
```

Destination Name—Syntax and Semantics

Slash & Dot Separators This aspect of the mapping between EMS destination names and SmartSockets subjects is straightforward, one-to-one, and bidirectional.

EMS destination names consist of tokens separated by the dot (.) character. SmartSockets subjects consists of tokens preceded by the slash (/) character (like UNIX directory pathnames).

For example, the EMS name `foo.bar.baz` corresponds to the SmartSockets name `/foo/bar/baz`. (Remember that SmartSockets names must begin with a leading slash, but EMS names need not begin with a leading dot. A leading dot indicates an empty element preceding it.)

The slash and dot characters have complementary roles in EMS and SmartSockets. In EMS slash is an ordinary character, while dot is a separator. In SmartSockets slash is a separator, while dot is an ordinary character. To translate names between EMS and SmartSockets, substitute these characters one for another. For example, the EMS name `foo/bar.baz` corresponds to the SmartSockets name `/foo.bar/baz`. However, to avoid confusion, we discourage using either slash or dot as ordinary characters.

Wildcard Star Although both EMS and SmartSockets both interpret the star (*) character as a wildcard, they differ in its semantics. In this aspect, the mapping is not one-to-one.

In EMS, star can match any whole token of a name, but not part of a token. In SmartSockets, star can match part of an token—for example, `/foo/b*/baz` matches `/foo/bar/baz` and `/foo/box/baz`.

If you are familiar with SmartSockets wildcards but not EMS wildcards, see [Wildcards on page 77](#).

Trailing Wildcard In EMS the greater-than (>) character is a wildcard that matches any number of trailing tokens. In SmartSockets a string of three dots (..) signifies identical semantics.

Topics

Topics can both export and import messages. Accordingly, you can configure topic definitions (in the configuration file `topics.conf`) with `import` and `export` properties that specify one or more external transports:

- | | |
|---------------------|---|
| <code>import</code> | <ul style="list-style-type: none"> <code>import</code> instructs <code>tibemsd</code> to import messages that arrive on those transports from SmartSockets, and deliver them to the EMS destination. |
| <code>export</code> | <ul style="list-style-type: none"> <code>export</code> instructs <code>tibemsd</code> to take messages that arrive on the EMS destination, and export them to SmartSockets via those transports. |



The EMS server *never* re-exports an imported message on the same topic.

(For general information about `topics.conf` syntax and semantics, see [topics.conf on page 257](#). You can also configure topics using the administration tool command `addprop topic`.)

Example

For example, the following `tibemsd` commands configure the topic `myTopics.news` to import and export messages on three transports.

```
addprop topic myTopics.news import="SS01,SS02"
addprop topic myTopics.news export="SS01,SS02,SS03"
```

SmartSockets messages with subject `/myTopics/news` arrive at `tibemsd` over the transports `SS01` and `SS02`. EMS clients can receive those messages by subscribing to `myTopics.news`.

EMS messages sent to `myTopics.news` are exported to SmartSockets over all three transports—`SS01`, `SS02` and `SS03`. SmartSockets clients of the corresponding RTservers can receive those messages by subscribing to `/myTopics/news`.

Import Only when Subscribers Exist

When a topic specifies `import` on a connected transport, `tibemsd` imports messages only when the topic has registered subscribers.

Wildcards

Wildcards in the `import` and `export` properties obey EMS syntax and semantics (not SmartSockets syntax and semantics); see [Destination Name—Syntax and Semantics on page 432](#).

Queues

Queues can import messages, but cannot export them.

Configuration

You can configure queue definitions (in the configuration file `queues.conf`) with the `import` property that specify one or more external transports.

- `import` instructs `tibemsd` to import messages that arrive on those transports from SmartSockets, and deliver them to the EMS destination.

(For general information about `queues.conf` syntax and semantics, see [queues.conf on page 250](#). You can also configure queues using the administration tool command `addprop queue`.)

Example For example, the following `tibemsdadmin` command configures the queue `myTopics.news` to import messages on the transports `SS01` and `SS02`.

```
addprop queue myQueue.in import="SS01,SS02"
```

SmartSockets messages with subject `/myQueue/in` arrive at `tibemsd` over the transports `SS01` and `SS02`. EMS clients can receive those messages by subscribing to `myQueue.in`.

Import—Start and Stop

When a queue specifies `import` on a connected transport, `tibemsd` immediately begins importing messages to the queue, even when no receivers exist for the queue.

For static queues (configured by an administrator) `tibemsd` continues importing until you explicitly delete the queue.

Wildcards

Wildcards in the `import` property obey EMS syntax and semantics (not SmartSockets syntax and semantics); see [Destination Name—Syntax and Semantics on page 432](#).

EMS clients cannot subscribe to wildcard queues—however, you can define wildcard queues in the EMS server for the purpose of property inheritance. That is, you can configure a static queue named `foo.*` and set properties on it, so that child queues named `foo.bar` and `foo.baz` will both inherit those properties.

If you define a queue that imports `foo.*`, `tibemsd` begins importing all matching messages from SmartSockets. As messages arrive, `tibemsd` creates dynamic child queues (for example, `foo.bar` and `foo.baz`) and delivers the messages to them. Notices that `tibemsd` delivers messages to these dynamic child queues even when no subscribers exist to drain them.

Import Issues

This section presents issues associated with importing messages to EMS from SmartSockets—whether on a topic or a queue.

Import Destination Names Must be Unique



When a topic and a queue share the same name, *at most one* of them may set the `import` property. For example, if a topic `foo.bar` and a queue `foo.bar` are both defined, only one may specify the `import` property.

JMSReplyTo

When `tibemsd` imports and translates a SmartSockets message, it sets the `JMSReplyTo` field of the EMS message to the value of the SmartSockets `reply_to` header, so that EMS clients can reply to the message.

Usually this value represents a SmartSockets subject. You must explicitly configure `tibemsd` to create a topic with a corresponding name, which exports messages to SmartSockets.

Guaranteed Delivery



For full end-to-end guaranteed delivery from SmartSockets to EMS, all three of these conditions must be true:

- SmartSockets senders must send messages with guaranteed message delivery (GMD).
- The transport definition must set `topic_import_dm` or `queue_import_dm` (as appropriate) to `TIBEMS_PERSISTENT`.
- A durable subscription for the EMS topic or queue must exist.

For export guarantees, see [Guaranteed Delivery on page 437](#).

Export Issues

This section presents issues associated with exporting messages from EMS to SmartSockets.

JMSReplyTo

Topics	Consider an EMS message in which the field <code>JMSReplyTo</code> contains a topic. When exporting such a message to SmartSockets, you must explicitly configure <code>tibemsd</code> to import replies from SmartSockets to that reply topic.
Temporary Topics	Consider an EMS message in which the field <code>JMSReplyTo</code> contains a temporary topic. When <code>tibemsd</code> exports such a message to SmartSockets, it <i>automatically</i> arranges to import replies to that temporary topic from SmartSockets; you do not need to configure it explicitly.

Wildcard Subscriptions

Star Wildcard	Both EMS and SmartSockets interpret the star character (*) as a wildcard—but with different semantics. EMS accepts star only as a whole element, which matches a whole element. In contrast, SmartSockets accepts star as part of an element, matching a substring within the element.
---------------	--

When a SmartSockets client subscribes to `foo.bar*`, then configure `tibemsd` to export the superset `foo.*`; RTserver narrows the set by delivering only messages that match subscribers. For a full discussion of the differences between EMS and SmartSockets wildcards, see [Destination Name—Syntax and Semantics on page 432](#).

Guaranteed Delivery



For full end-to-end guaranteed delivery to SmartSockets from EMS, both of these conditions must be true:

- EMS senders must send persistent messages.
- The transport definition must set `delivery_mode` to `gmd_some` or `gmd_all` (as appropriate).

To preserve undelivered GMD messages in the EMS undelivered queue, see [preserve_gmd on page 430](#). For import guarantees, see [Guaranteed Delivery on page 436](#).

Message Translation

JMS Header Fields

EMS supports the 11 predefined JMS header fields; see [JMS Message Header Fields on page 17](#).

Two Special Cases	<p>These two header fields are special cases:</p> <ul style="list-style-type: none"> • JMS header <code>JMSDestination</code> corresponds to SmartSockets <code>dest</code>. • JMS header <code>JMSReplyTo</code> corresponds to SmartSockets <code>reply_to</code>.
Import	When importing a SmartSockets message to an EMS message, <code>tibemsd</code> does not set any JMS header fields, except for the special cases noted above.
Export	<p>When exporting an EMS message to a SmartSockets message, <code>tibemsd</code> groups all the JMS header fields (except for the special cases noted above) into a single submessage within the SmartSockets message. The field <code>JMSHeaders</code> contains that submessage. Fields of the submessage map the names of JMS header fields to their values.</p> <p><code>tibemsd</code> ignores any JMS header fields that are null or absent—it omits them from the exported message.</p> <p>You can instruct <code>tibemsd</code> to suppress the entire header submessage in all exported messages by setting the transport property <code>export_headers = false</code>.</p>

JMS Property Fields

Import	<p>When importing a SmartSockets message to an EMS message, <code>tibemsd</code> sets these JMS properties:</p> <ul style="list-style-type: none"> • <code>JMS_TIBCO_IMPORTED</code> gets the value <code>true</code>, indicating that the message did not originate from an EMS client. • <code>JMS_TIBCO_MSG_EXT</code> gets the value <code>true</code>, indicating that the message <i>might</i> contain submessage fields or array fields. • <code>JMS_TIBCO_SS_SENDER</code> gets the value of the SmartSockets sender header field (in SmartSockets syntax). <p>In addition, <code>tibemsd</code> maps SmartSockets message properties to EMS properties; for details see SmartSockets Message Properties on page 439.</p>
--------	--

- Export** When exporting an EMS message to a SmartSockets message, `tibemsd` groups all the JMS property fields into a single submessage within the SmartSockets message. The field `JMSProperties` contains that submessage. Fields of the submessage map the names of JMS property fields to their values.
- `tibemsd` ignores any JMS property fields that are not set, or are set to null—it omits them from the exported message.
- You can instruct `tibemsd` to suppress the entire properties submessage in the exported message by setting the transport property `export_properties = false`.

SmartSockets Message Properties

In release 4.1.0 (and later), `tibemsd` maps SmartSockets message headers to EMS message properties on import. [Table 69](#) summarizes the mapping. The first column indicates the EMS property, and the second column indicates the SmartSockets method that gets the corresponding header.

- Import** The transport parameter `import_ss_headers` governs the import behavior. The third column of [Table 69](#) lists the values of that parameter for which `tibemsd` imports the message property in that row. See [import_ss_headers on page 429](#).
- Export** EMS client programs may modify the values of these properties within imported messages for re-export to SmartSockets. (However, exporting a native EMS message does not carry these properties to SmartSockets.)
- Export of these properties depends on the value of the transport parameter [export_properties on page 431](#).
- When exporting an EMS message to SmartSockets, `tibemsd` maps these properties in reverse. In most cases, the mapping is symmetric—export maps them back to the same SmartSockets header. However, three exceptions (`JMS_TIBCO_SS_SENDER`, `JMS_TIBCO_SS_MESSAGE_ID` and `JMS_TIBCO_SS_SEQ_NUM`) are asymmetric—export maps them to subfields of the field `JMSProperties` within the SmartSockets message. The fourth column of [Table 69](#) indicates this asymmetry.

Table 69 SmartSockets Mapping Message Properties (Import & Export) (Sheet 1 of 2)

EMS Property	SmartSockets Method	Import	Export Asymmetr.
<code>JMS_TIBCO_SS_SENDER</code>	<code>TipcMsgGetSender</code>	<code>none</code> <code>type_num</code> <code>all</code>	Asymmetr.

Table 69 SmartSockets Mapping Message Properties (Import & Export) (Sheet 2 of 2)

EMS Property	SmartSockets Method	Import	Export Asymmetr.
JMS_TIBCO_SS_TYPE_NUM	TipcMsgGetType	type_num all	
JMS_TIBCO_SS_DELIVERY_MODE	TipcMsgGetDeliveryMode	all	
JMS_TIBCO_SS_LB_MODE	TipcMsgGetLbMode	all	
JMS_TIBCO_SS_EXPIRATION	TipcMsgGetExpiration	all	
JMS_TIBCO_SS_PRIORITY	TipcMsgGetPriority	all	
JMS_TIBCO_SS_SENDER_TIMESTAMP	TipcMsgGetSenderTimestamp	all	
JMS_TIBCO_SS_CORRELATION_ID	TipcMsgGetCorrelationId	all	
JMS_TIBCO_SS_USER_PROP	TipcMsgGetUserProp	all	
JMS_TIBCO_SS_MESSAGE_ID	TipcMsgGetMessageId	all	Asymmetr.
JMS_TIBCO_SS_SEQ_NUM	TipcMsgGetSeqNum	all	Asymmetr.

Message Body

tibemspd can export messages with any JMS message body type to TIBCO SmartSockets. Conversely, tibemspd can import messages with any message type from TIBCO SmartSockets.

For information about JMS body types, see [JMS Message Bodies on page 22](#).

For information about the structure of messages, see [JMS Message Structure on page 17](#).

- Import
- When importing a SmartSockets message, tibemspd translates it to one of two EMS message body types:

 - If the SmartSockets message contains only *unnamed* fields, then it translates into a JMSStreamMessage. The stream contains the values of the unnamed fields in the same order as they appear in the SmartSockets message.
 - If the SmartSockets message contains one or more named fields, then it translates into a JMSMapMessage. The map message contains the named fields; the order of the fields is indeterminate.

Export When exporting an EMS message, tibemsd translates it to one of six SmartSockets message types (see [Table 70](#)) with the following structure:

- The named field `JMSHeaders` is the first field (omitted when the transport parameter `export_headers` is false). It contains a submessage; see [JMS Header Fields on page 438](#).
- The named field `JMSProperties` is the next field (omitted when the transport parameter `export_properties` is false). It contains a submessage; see [JMS Property Fields on page 438](#).
- The data fields follow the JMS headers and properties (when present). For details about field names and types, see the third column of [Table 70](#).

Table 70 SmartSockets: Mapping Message Types (Export)

JMS Message Type	SmartSockets Message Type	Data Fields
JMSBytesMessage	T_MT_JMS_BYTES	One unnamed field of type T_MSG_FT_BINARY
JMSMapMessage	T_MT_JMS_MAP	Named fields; indeterminate order
JMSObjectMessage	T_MT_JMS_OBJECT	One unnamed field of type T_MSG_FT_BINARY
JMSStreamMessage	T_MT_JMS_STREAM	Unnamed fields in order
JMSTextMessage	T_MT_JMS_TEXT	One unnamed field of type T_MSG_FT_STR
All other JMS message types	T_MT_INFO	No data fields

Data Types

[Table 71](#) presents the mapping between EMS datatypes and SmartSockets datatypes. The mapping is bidirectional, except for a few SmartSockets types that have no corresponding EMS type (for these types the mapping is marked as unidirectional in the middle column of [Table 71](#)).

Table 71 SmartSockets: Mapping Data Types (Sheet 1 of 2)

EMS	Map	SmartSockets
Boolean		T_MSG_FT_BOOL
Byte		T_MSG_FT_BYTE
Character		T_MSG_FT_CHAR
Short		T_MSG_FT_INT2
Integer		T_MSG_FT_INT4
Long		T_MSG_FT_INT8
Float		T_MSG_FT_REAL4
Double		T_MSG_FT_REAL8
Double	<—	T_MSG_FT_TIMESTAMP
String		T_MSG_FT_STR
String	<—	T_MSG_FT_XML
String	<—	T_MSG_FT_UTF8
Byte Array		T_MSG_FT_BINARY
Short Array	<—	T_MSG_FT_BOOL_ARRAY
Short Array		T_MSG_FT_INT2_ARRAY
Integer Array		T_MSG_FT_INT4_ARRAY
Long Array		T_MSG_FT_INT8_ARRAY
Float Array		T_MSG_FT_REAL4_ARRAY
Double Array		T_MSG_FT_REAL8_ARRAY

Table 71 SmartSockets: Mapping Data Types (Sheet 2 of 2)

EMS	Map	SmartSockets
Double Array	<—	T_MSG_FT_TIMESTAMP_ARRAY
Stream Message		T_MSG_FT_MSG
Map Message		(See Import on page 440.)

Destination Names

tibemsd automatically translates destination names when importing or exporting a message; see [Slash & Dot Separators on page 432](#).

When importing, it translates names in the SmartSockets subject and reply_to fields. When exporting, it translates names in the EMS JMSDestination and JMSReplyTo fields.

Chapter 17 **Monitoring Server Activity**

System administrators must monitor and manage the TIBCO Enterprise Message Service server. The logging, monitoring, and statistics facilities provided by the server allow system administrators to effectively view system activity and track system performance.

Topics

- [Log Files and Tracing, page 446](#)
- [Message Tracing, page 452](#)
- [Monitoring Server Events, page 454](#)
- [Working with Server Statistics, page 460](#)

Log Files and Tracing

You can configure the TIBCO Enterprise Message Service server to write a variety of information to the log. Several parameters and commands control where the log is located as well as what information is written to the log. The log can be written to a file, to the system console, or to both.

Configuring the Log File

The `logfile` configuration parameter in `tibemsd.conf` controls the location and the name of the log file.

You can specify that the log file should be backed up and emptied after it reaches a maximum size. This allows you to rotate the log file and ensure that the log file does not grow boundlessly. The `logfile_max_size` configuration parameter allows you to specify the maximum size of the current log file. Set the parameter to 0 to specify no limit. Use KB, MB, or GB units.

Once the log file reaches its maximum size, it is copied to a file with the same name as the current log file except a sequence number is appended to the name of the backup file. On startup—and only on startup—the server queries the directory and determines the first available sequence number. It then uses the next sequence number when it needs to back up the current log file. By doing so, you can keep a continuous sequencing, as long as you retain the most recent log file (highest sequence number) between server restarts. Conversely, if you move or remove all log files before a server restart, then the sequencing will restart at 1.

For example, if the current log file is named `tibems.log`, the first copy is named `tibems.log.1`, the second is named `tibems.log.2`, and so on. Similarly, if the highest sequence number in use when the server starts is 19, or `tibemsd.log.19`, then the next backup file created will be named `tibemsd.log.20`. This is true even if you removed `tibemsd.log.19` and all other log files after the server started.

If `logfile_max_count` is specified, the server keeps at most the number of log files specified by that parameter, including the current log file. When the maximum number of log files has been reached and the server needs to back up the current log file, it deletes the oldest log file (the ones with smallest number). If you change the parameter setting, after the server is restarted, the next time it needs to rotate the log file it deletes however many of the lowest sequence numbered files required to reach the `logfile_max_count` maximum.

You can also dynamically force the log file to be backed up and truncated using the `rotatelog` command in `tibemsdadmin`. See [Command Listing on page 128](#) for more information about the `rotatelog` command.

For other configuration parameters that affect the log file, see [Tracing and Log File Parameters on page 220](#).

Tracing on the Server

The TIBCO Enterprise Message Service server can be configured to produce tracing messages. These messages can describe actions performed for various areas of functionality (for example, Access Control, Administration, or Routing). These messages can also provide information about activities performed on or by the server, or the messages can provide warnings in the event of failures or illegal actions.

Trace messages can be sent to a log file, the console, or both. You configure tracing in the following ways:

- By configuring the `log_trace` and/or `console_trace` parameters in the `tibemsd.conf` file; see [Table 16 on page 143](#).
- By specifying the `-trace` option when starting the server
- By using the `set server` command when the server is running.

`log_trace` and `console_trace` can be used to configure what types of messages are to go to the log file and to the console.



When you want trace messages to be sent to a log file, you must also configure the `logfile` configuration parameter. If you specify `log_trace`, and the `logfile` configuration parameter is not set to a valid file, the tracing options are stored, but they are not used until the server is started with a valid log file.

When configuring log or console tracing, you have a variety of options for the types of trace messages that can be generated. [Table 72 on page 448](#) describes the available tracing options.

Table 72 Server Tracing Options

Trace Option	Description
DEFAULT	Sets the trace options to the default set. This includes: <ul style="list-style-type: none">• INFO• WARNING• ACL• LIMITS• ROUTE• ADMIN• RVADV• CONNECT_ERROR• CONFIG• MSG
ACL	Prints a message when a user attempts to perform an unauthorized action. For example, if the user attempts to publish a message to a secure topic for which the user has not been granted the publish permission.
ADMIN	Prints a message whenever an administration function is performed.
AUTH	Prints a message when the server authenticates a user using an external LDAP system.
CONFIG	Prints information about configuration files and their contents as the EMS server is starting up.
CONNECT	Prints a message when a user attempts to connect to the server.
CONNECT_ERROR	Prints a message when an error occurs on a connection.
DBSTORE	Prints a message when a database store is created, along with general database store information and errors.
DEST	Prints a message when a dynamic destination is created.

Table 72 Server Tracing Options

Trace Option	Description
FLOW	Prints a message when the server enforces flow control or stops enforcing flow control on a destination.
INFO	Prints messages as the server performs various internal housekeeping functions, such as creating a configuration file, opening the persistent database files, and purging messages. Also prints a message when tracking by message ID is enabled or disabled.
JAAS	<p>Prints messages related to any extensible security modules.</p> <p>Messages are printed when a username and password are passed to the LoginModule for authentication, and when a user and action are passed to the Permissions Modules for authorization.</p>
JVM	Prints startup information about the JVM configuration, as well as any output from custom modules running in the JVM that uses <code>System.out</code> .
JVMERR	Prints output from custom modules running in the JVM that uses <code>System.err</code> .
LDAP_DEBUG	Prints messages when LDAP is used for authentication or to obtain group information.
LIMITS	Prints a message when a limit is exceeded, such as the maximum size for a destination.
LOAD	Prints the paths of any dynamically loaded libraries. For example, the <code>tibemsd</code> can load <code>Zlib</code> , <code>SmartSockets</code> , and <code>SSL</code> libraries.
MEMORY	Prints a server trace information when reserve memory is triggered because of low server memory conditions.

Table 72 Server Tracing Options

Trace Option	Description
MSG	Specifies that message trace messages should be printed. Message tracing is enabled/disabled on a destination or on an individual message. If message tracing is not enabled for any messages or destinations, no trace messages are printed when this option is specified for log or console tracing. See Message Tracing on page 452 for more information about message tracing.
MULTICAST	Prints a message when a message consumer subscribes or attempts to subscribe to a multicast-enabled topic, along with general multicast information and errors.
PRODCONS	Prints a message when a client creates or closes a producer or consumer.
ROUTE	Prints a message when routes are created or when a route connection is established.
ROUTE_DEBUG	Prints status and error messages related to the route.
RVADV	Prints TIBCO Rendezvous advisory messages whenever they are received.
SS	Prints trace messages related to SmartSockets bridges.
SSL	Prints detailed messages of the SSL process, including certificate content.
SSL_DEBUG	Prints messages that trace the establishment of SSL connections.
TX	Prints a message when a client performs a transaction.
WARNING	Prints a message when a failure of some sort occurs, usually because the user attempts to do something illegal. For example, a message is printed when a user attempts to publish to a wildcard destination name.

Specify tracing with a comma-separated list of trace options. You may specify trace options in three forms:

- plain A trace option without a prefix character replaces any existing trace options.

- + A trace option preceded by + adds the option to the current set of trace options.
- - A trace option preceded by - removes the option from the current set of trace options.

Examples

The following example sets the trace log to only show messages about access control violations.

```
log_trace=ACL
```

The next example sets the trace log to show all default trace messages, in addition to SSL messages, but ADMIN messages are not shown.

```
log_trace=DEFAULT, -ADMIN, +SSL
```

The next example sends a trace message to the console when a TIBCO Rendezvous advisory message arrives.

```
console_trace=RVADV
```

Message Tracing

In addition to other server activity, you can trace messages as they are processed. Trace entries for messages are only generated for destinations or messages that specify tracing should be performed. For destinations, you specify the `trace` property to enable the generation of trace messages. For individual messages, the `JMS_TIBCO_MSG_TRACE` property specifies that tracing should be performed for this message, regardless of the destination settings. The sections below describe the tracing properties for destinations and messages.

Message trace entries can be output to either the console or the log. The `MSG` trace option specifies that message trace entries should be displayed, and the `DEFAULT` trace option includes the `MSG` option. See [Tracing on the Server on page 447](#) for more information about specifying trace options.

You must set the tracing property on either destinations or messages and also set the `MSG` or `DEFAULT` trace option on the console or the log before you can view trace entries for messages.



EMS tracing features do not filter unprintable characters from trace output. If your application uses unprintable characters within messages (whether in data or headers), the results of message tracing are unpredictable.

Enabling Message Tracing for a Destination

The `trace` property on a destination specifies that trace entries are generated for that destination.

The `trace` property can optionally be specified as `trace=body`. Setting `trace=body` includes the message body in trace messages. The EMS server prints up to one kilobyte of a message string field, and up to a total message size of 8 KB. The trace message indicates if the full message is not printed.

Setting `trace` without the `body` option specifies that only the message sequence and message ID are included in the trace message.

When message tracing is enabled for a destination, a trace entry is output for each of the following events that occur in message processing:

- messages are received into a destination
- messages are sent to consumers
- messages are imported or exported to/from an external system
- messages are acknowledged
- messages are sent across a destination bridge

- messages are routed

Replies to request messages are traced only when the reply destination has the trace property. Similarly, replies to exported messages are only traced when the trace property is set.

Enabling Message Tracing on a Message

You can enable tracing on individual messages by setting the `JMS_TIBCO_MSG_TRACE` property on the message. The value of the property can be either `null` (Java/.NET `null` or `NULL` in C) or the string `"body"`. Setting the property to `null` specifies only the message ID and message sequence will be included in the trace entries for the message. Setting the property to `"body"` specifies the message body will be included in the trace entries for the message.

When the `JMS_TIBCO_MSG_TRACE` property is set for a message, trace entries are generated for the message as it is processed, regardless of whether the trace property is set for any destinations the message passes through. Trace messages are generated for the message when it is sent by the producer and when it is received by the consumer.

Monitoring Server Events

The TIBCO Enterprise Message Service server can publish topic messages for internal system events. For example, the server can publish a message when users connect or disconnect. System event messages contain detail about the event stored in properties of the message. This section gives an overview of the monitoring facilities provided by the server. For a list of monitor topics and a description of the message properties for each topic, see [Appendix A, Monitor Messages](#), on page 533.

System Monitor Topics

The TIBCO Enterprise Message Service server can publish messages to various topics when certain events occur. There are several types of event classes, each class groups a set of related events. For example, some event classes are connection, admin, and route. Each event class is further subdivided into the events for each class. For example, the connection class has two events: connect and disconnect. These event classes are used to group the system events into meaningful categories.

All system event topic names begin with `$sys.monitor`. The remainder of the name is the event class followed by the event. For example, the server publishes a message to the topic `$sys.monitor.connection.disconnect` whenever a client disconnects from the server. The naming scheme for system event topics allows you to create wildcard subscriptions for all events of a certain class. For example, to receive messages whenever clients connect or disconnect, you would create a topic subscriber for the topic `$sys.monitor.connection.*`.

Monitor topics are created and maintained by the server. Monitor topics are not listed in the `topics.conf` file. Users can subscribe to monitor topics but cannot create them.

Monitoring Messages

You can monitor messages processed by a destination as they are sent, received, or acknowledged. You can also monitor messages that have prematurely exited due to expiration, being discarded, or a [maxRedelivery](#) failure.

The `$sys.monitor` topic for monitoring messages has the following format:

```
$sys.monitor.D.E.destinationName
```


Where *D* is the type of destination, *E* is the event you wish to monitor, and *destinationName* is the name of the destination whose messages you wish to monitor. [Table 73](#) describes the possible values of *D* and *E* in message monitoring topics.

Table 73 Message monitoring qualifiers (Sheet 1 of 2)

Qualifier	Value	Description
D	T	Destination to monitor is a topic. Include the message body in the monitor message as a byte array. Use the <code>createFromBytes()</code> method when viewing the monitor message to recreate the message body, if desired.
	t	Destination to monitor is a topic. Do not include the message body in the monitor message.
	Q	Destination to monitor is a queue. Include the message body in the monitor message as a byte array. Use the <code>createFromBytes()</code> method when viewing the monitor message to recreate the message body, if desired.
	q	Destination to monitor is a queue. Do not include the message body in the monitor message.

Table 73 Message monitoring qualifiers (Sheet 2 of 2)

Qualifier	Value	Description
E	s	Monitor message is generated when a message is sent by the server to: <ul style="list-style-type: none">• a consumer• a route• an external system by way of a transport
r		Monitor message is generated when a message is received by the specified destination. This occurs when the message is: <ul style="list-style-type: none">• Sent by a producer• Sent by a route• Forwarded from another destination by way of a bridge• Imported from transport to an external system
a		Monitor message is generated when a message is acknowledged.
p		Monitor message is generated when a message prematurely exits due to expiration, being discarded, or a maxRedelivery failure.
*		Monitor message is generated when a message is sent, received, or acknowledged for the specified destination.

For example, `$sys.monitor.T.r.corp.News` is the topic for monitoring any received messages to the topic named `corp.News`. The message body of any received message is included in monitor messages on this topic. The topic `$sys.monitor.q.*.corp.*` monitors all message events (send, receive, acknowledge) for all queues matching the name `corp.*`. The message body is not included in this topic's messages.

The messages sent to this type of monitor topic include a description of the event, information about where the message came from (a producer, route, external system, and so on), and optionally the message body, depending upon the value of *D*. See [Appendix A, Monitor Messages, on page 533](#) for a complete description of the properties available in monitoring messages.

You must explicitly subscribe to a message monitoring topic. That is, subscribing to `$sys.monitor.>` will subscribe to all topics beginning with `$sys.monitor`, but it does not subscribe you to any specific message monitoring topic such as `$sys.monitor.T.*.foo.bar`. However, if another subscriber generates interest in the message monitor topics, this subscriber will also receive those messages.

You can specify wildcards in the *destinationName* portion of the message monitoring topic to subscribe to the message monitoring topic for all matching destinations. For example, you can subscribe to `$sys.monitor.T.r.>` to monitor all messages received by all topics. For performance reasons, you may want to avoid subscribing to too many message monitoring topics. See [Performance Implications of Monitor Topics on page 458](#) for more information.

Viewing Monitor Topics

Monitor topics are similar to other topics. To view these topics, create a client application that subscribes to the desired topics.

Because monitor topics contain potentially sensitive system information, authentication and permissions are always checked when clients access a monitor topic. That is, even if authentication for the server is disabled, clients are not able to access monitor topics unless they have logged in with a valid username and password and the user has permission to view the desired topic.

The `admin` user and members of the `$admin` group have permission to perform any server action, including subscribing to monitor topics. All other users must be explicitly granted permission to view monitor topics before the user can successfully create subscribers for monitor topics. For example, if user BOB is not a member of the `$admin` group, and you wish to allow user BOB to monitor all connection events, you can grant BOB the required permission with the following command using the administration tool:

```
grant topic $sys.monitor.connection.* BOB subscribe
```

Bob's application can then create a topic subscriber for `$sys.monitor.connect.*` and view any connect or disconnect events.



Topics starting with `$sys.monitor` do not participate in any permission inheritance from parent topics other than those starting with `$sys.monitor` (that is, `.*` or `.*>` is not a parent of `$sys.monitor`).

Therefore, granting permission to a user to subscribe to `>` does not allow that user to subscribe to `$sys.monitor` topics. You must explicitly grant users permission to `$sys.monitor` topics (or parent topics, such as `$sys.monitor.admin.*`) for a user to be able to subscribe to that topic.

Monitor topics publish messages of type `MapMessage`. Information about the event is stored within properties in the message. Each system event has different properties. [Appendix A, Monitor Messages, on page 533](#) describes each of the monitor topics and the message properties for the messages published on that topic. Your application can receive and display all or part of a monitor message, just as it would handle any message sent to a topic. However, there are some ways in which monitor messages are handled differently from standard messages:

- Monitor messages cannot be routed to other servers.
- Monitor messages are not stored persistently on disk.
- Monitor messages are not swapped from process memory to disk.

You can have any number of applications that subscribe to monitor messages. You can create different applications that subscribe to different monitor topics, or you can create one application that subscribes to all desired monitor topics. Your topic subscribers can also use message selectors to filter the monitor messages so your application receives only the messages it is interested in.

Performance Implications of Monitor Topics

The TIBCO Enterprise Message Service server only generates messages for monitor topics that currently have subscribers. So, if no applications subscribe to monitor topics, no monitor messages are generated. Generating a monitor message does consume system resources, and therefore you should consider what kinds of monitoring your environment requires. System performance is affected by the number of subscribers for monitor topics as well as the frequency of messages for those topics.

For development and testing systems, monitoring all system events is probably desirable. Usually, development and testing systems do not have large message volumes, and monitoring can give you information about system problems.

For production systems, monitoring all events may have an adverse effect on system performance. Therefore, you should not create topic subscribers for `$sys.monitor.>` in your production system. Also, monitor events are likely to be added in future releases, so the number of monitor topics may grow. Subscriptions to monitor topics in production systems should always be limited to specific monitor topics or wildcard subscriptions to specific classes of monitor topics that are required.

Also, consider the frequency of messages to each monitor topic. System administration events, such as creating topics, routes, and changing permissions, do not occur frequently, so creating subscriptions for these types of events will most likely not have a significant effect on performance.

Also, using message selectors to limit monitor messages can improve performance slightly. The server does not send any messages that do not match a subscriber's message selector. Even though the message is not sent, the message is still generated. Therefore there is still system overhead for subscribers to a monitor topic, even if all messages for that topic do not match any subscriber's message selector filter.

Working with Server Statistics

The TIBCO Enterprise Message Service server allows you to track incoming and outgoing message volume, message size, and message statistics for the server overall as well as for each producer, consumer, or route. You can configure the type of statistics collected, the interval for computing averages, and amount of detail for each type.

Statistic tracking can be set in the server's configuration file, or you can change the configuration dynamically using commands in the administration tool or by creating your own application with the administration APIs.

Statistics can be viewed using the administration tool, or you can create your own application that performs more advanced analysis of statistics using the administration APIs.

This section details how to configure and view statistics using the configuration files and administration tool commands. For more information about the administration APIs, see the description of `com.tibco.tibjms.admin` in the online documentation.



The TIBCO Enterprise Message Service server tracks the number of incoming or outgoing messages, but only messages sent or received by a producer, consumer, or route are tracked. The server also sends system messages, but these are not included in the number of messages.

However, the server can add a small amount of data to a message for internal use by the server. This overhead is counted in the total message size, and you may notice that some messages have a greater message size than expected.

Overall Server Statistics

The server always collects certain overall server statistics. This includes the rate of inbound and outbound messages (expressed as number of messages per second), message memory usage, disk storage usage, and the number of destinations, connections, and durable subscriptions. Gathering this information consumes virtually no system resources, therefore these statistics are always available. You can view overall server statistics by executing the `show server` command.

The default interval for collecting overall server statistics is 1 second. You may wish to view average system usage statistics over a larger interval. The `server_rate_interval` configuration parameter controls the collection interval for server statistics. The parameter can be set in the configuration file or dynamically using the `set server` command. This parameter can only be set to positive integers greater than zero.

Enabling Statistic Gathering

Each producer, consumer, destination, and route can gather overall statistics and statistics for each of its destinations. To enable statistic gathering, you must set the `statistics` parameter to `enabled`. This parameter can be specified in the configuration file, and it can be changed dynamically using the `set server` command.

The `statistics` parameter allows you to globally enable and disable statistic gathering. Statistics are kept in server memory for the life of each object. If you wish to reset the total statistics for all objects to zero, disable statistic gathering, then re-enable it. Server statistics are also reset when the server shuts down and restarts, or in the event of a fault-tolerant failover.

For each producer, consumer, destination, and route the total number of sent/received messages and total size of messages is maintained. Also, producers and consumers keep these statistics for each destination that they use to send or receive messages.

The rate of incoming/outgoing messages and message size is calculated over an interval. By default, the average is calculated every 3 seconds. You can increase or decrease this value by altering the `rate_interval` parameter. This parameter can be set in the configuration file or dynamically using the `set server` command. Setting this parameter to 0 disables the tracking of statistics over an interval—only the total statistics for the destination, route, producer, or consumer are kept.

Gathering total statistics for producers, consumers, destinations, and routes consumes few system resources. Under most circumstances, enabling statistic gathering and average calculations should not affect system performance.

Detailed Statistics

In some situations, the default statistic gathering may not be sufficient. For example, if a topic subscriber subscribes to wildcard topics, the total statistics for all topics that match the wildcard are kept. You may wish to get further detail in this case and track the statistics for each actual topic the subscriber receives.

The following situations may require detailed statistic gathering:

- Topic subscribers that subscribe to wildcard topics
- Message producers that do not specify a destination when they are created. These message producers can produce messages for any destination, and the destination name is specified when a message is sent.
- Routes can have incoming and outgoing messages on many different topics.
- Channels can also have outgoing messages on many different topics.

To enable detailed statistics, set the `detailed_statistics` parameter to the type of statistics you wish to receive. The parameter can have the following values:

- `NONE` — disables detailed statistic gathering.
- `CONSUMERS` — enables detailed statistics for topic subscribers with wildcard topic names.
- `PRODUCERS` — enables detailed statistics for producers that do not specify a destination when they are created.
- `ROUTES` — enables detailed statistics for routes.
- `CHANNELS` — enables detailed statistics for channels.

You can set the `detailed_statistics` parameter to `NONE` or any combination of `CONSUMERS`, `PRODUCERS`, `ROUTES`, or `CHANNELS`. To specify more than one type of detailed statistic gathering, provide a comma-separated list of values. You can set the `detailed_statistics` parameter in the configuration file or dynamically by using the `set server` command. For example, the following `set server` command enables detailed statistic tracking for producers and routes.

```
set server detailed_statistics = PRODUCERS,ROUTES
```

Collecting detailed statistics does consume memory, and can adversely affect performance when gathering a high volume of statistics. There are two parameters that allow you to control resource consumption when collecting detailed statistics. First, you can control the amount of time statistics are kept, and second you can set a maximum amount of memory for detailed statistic gathering. When application programs create many dynamic destinations, we recommend against gathering detailed statistics.

The `statistics_cleanup_interval` parameter controls how long detailed statistics are kept. This parameter can be set either in the configuration file or dynamically with the `set server` command. By default, statistics are kept for 15 seconds. For example, if there is a topic subscriber for the topic `foo.*`, and the subscriber receives a message on topic `foo.bar`, if no new messages arrive for topic `foo.bar` within 15 seconds, statistics for topic `foo.bar` are deleted for that consumer. You can set this parameter to 0 to signify that all detailed statistics are to be kept indefinitely. Of course, statistics for an object only exist as long as the object itself exists. That is, if a message consumer terminates, all detailed statistics for that consumer are deleted from memory.

The `max_stat_memory` parameter controls the amount of memory used by detailed statistics. This parameter can be set either in the configuration file or dynamically with the `set server` command. By default, this parameter is set to 0 which signifies that detailed statistics have no memory limit. If no units are specified, the value of this parameter is in bytes. Optionally, you can specify units

as KB, MB, or GB. When the specified limit is reached, the server stops collecting new statistics. The server will only resume collecting statistics if the amount of memory used decreases (for example, if the `statistics_cleanup_interval` is set and old statistics are removed).

Displaying Statistics

When statistic collecting is enabled, you can view statistics for producers, consumers, routes, and destinations using the `show stat` command in the administration tool.

The `show stat` command allows you to filter the statistics based on destination name, user name, connection ID, or any combination of criteria. You can optionally specify the `total` keyword to retrieve only the total statistics (this suppresses the detailed output). You can also optionally specify the "wide" keyword when displaying statistics for destinations or routes. This specifies that inbound and outbound message statistics should be displayed on the same line (the line can be 100 characters or more).

The following illustrates displaying statistics for a route where detailed statistic tracking is enabled.

```
tcp://server1:7322> show stat route B
Inbound statistics for route 'B':
```

Destination	Total Count		Rate/Second	
	Msgs	Size	Msgs	Size
<total>	189	37.9 Kb	10	2.0 Kb
Topic: dynamic.0	38	7.6 Kb	2	0.4 Kb
Topic: dynamic.1	38	7.6 Kb	2	0.4 Kb
Topic: dynamic.2	38	7.6 Kb	2	0.4 Kb
Topic: dynamic.3	38	7.6 Kb	2	0.4 Kb
Topic: dynamic.4	37	7.4 Kb	2	0.4 Kb

```
Outbound statistics for route 'B':
```

Destination	Total Count		Rate/Second	
	Msgs	Size	Msgs	Size
<total>	9538	1.9 MB	10	2.1 Kb
Topic: dynamic.0	1909	394.9 Kb	2	0.4 Kb
Topic: dynamic.1	1908	394.7 Kb	2	0.4 Kb
Topic: dynamic.2	1907	394.5 Kb	2	0.4 Kb
Topic: dynamic.3	1907	394.5 Kb	2	0.4 Kb
Topic: dynamic.4	1907	394.5 Kb	2	0.5 Kb

See [show stat on page 168](#) for more information and detailed syntax of the `show stat` command.

Chapter 18 Using the SSL Protocol

Secure Sockets Layer (SSL) is a protocol that provides secure authentication and transmits encrypted data over the Internet or an internal network. Most web browsers support SSL, and many Web sites and Java applications use it to obtain confidential user information, such as credit card numbers.

The SSL protocol is complex, and this chapter is not a complete description of SSL. Instead, this chapter describes how to configure SSL in the TIBCO Enterprise Message Service server and in client applications that communicate with the server. For a more complete description of SSL, see the SSL specification at <http://www.mozilla.org/projects/security/pki/nss/ssl/>.

Topics

- [SSL Support in TIBCO Enterprise Message Service, page 466](#)
- [Digital Certificates, page 467](#)
- [File Names for Certificates and Keys, page 469](#)
- [Configuring SSL in the Server, page 471](#)
- [Configuring SSL in EMS Clients, page 472](#)
- [Specifying Cipher Suites, page 476](#)
- [SSL Authentication Only, page 482](#)
- [Enabling FIPS Compliance, page 483](#)

SSL Support in TIBCO Enterprise Message Service

TIBCO Enterprise Message Service supports the Secure Sockets Layer (SSL) protocol. SSL uses public and private keys to encrypt data over a network connection to secure communication between pairs of components:

- between an EMS client and the `tibemsd` server
- between the `tibemsadmin` tool and the `tibemsd` server
- between two routed servers
- between two fault-tolerant servers

SSL provides secure communication that works with other mechanisms for authentication available in the EMS server. When `authorization` is enabled in the server, the connection undergoes a two-phase authentication process. First, an SSL hand-shake between client and server initializes a secure connection. Second, the EMS server checks the credentials of the client using the supplied username and password. If the connecting client does not supply a valid username and password combination, the connection fails, even if the SSL handshake succeeded.



When authorization is enabled, usernames and passwords are always checked, even on SSL secured connections.

Implementations

The TIBCO Enterprise Message Service server and the C client libraries use OpenSSL for SSL support. For more information, see www.openssl.org.

EMS Java clients can use either JSSE (from Sun JavaSoft) or the SSL implementation from Entrust. The EMS Java installation includes JSSE; if you prefer to use Entrust, you must purchase and install the Entrust SSL implementation separately.

EMS .NET 2.0 clients use the Microsoft implementation of SSL. The Microsoft implementation of SSL is compatible with OpenSSL. Certificates required by the client can either be stored in files or the Microsoft certificate store. However, Microsoft requires that the root certificate be installed in the Microsoft Certificate Store, even when certificate files are in use.

EMS distributions usually build and include the latest versions of OpenSSL and OpenLDAP publicly available at the time of release. For exact version numbers see the Third Party Software License Agreements documented in the TIBCO Software Inc. End User License Agreement for TIBCO Enterprise Message Service.

Digital Certificates

Digital certificates are data structures that represent identities. EMS uses certificates to verify the identities of servers and clients. Though it is not necessary to validate either the server or the client for them to exchange data over SSL, certificates provide an additional level of security.

A digital certificate is issued either by a trusted third-party certificate authority, or by a security officer within your enterprise. Usually, each user and server on the network requires a unique digital certificate, to ensure that data is sent from and received by the correct party.

In order to support SSL, the EMS server must have a digital certificate. Optionally, EMS clients may also be issued certificates. If the server is configured to verify client certificates, a client must have a certificate and have it verified by the server. Similarly, an EMS client can be configured to verify the server's certificate. Once the identity of the server and/or client has been verified, encrypted data can be transferred over SSL between the clients and server.

A digital certificate has two parts—a public part, which identifies its owner (a user or server); and a private key, which the owner keeps confidential.

The public part of a digital certificate includes a variety of information, such as the following:

- The name of the owner, and other information required to confirm the unique identity of the subject. This information can include the URL of the web server using the digital certificate, or an email address.
- The subject's public key.
- The name of the certificate authority (CA) that issued the digital certificate.
- A serial number.
- The length of time the certificate will remain valid—defined by a start date and an end date.

The most widely-used standard for digital certificates is ITU-T X.509. TIBCO Enterprise Message Service supports digital certificates that comply with X.509 version 3 (X.509v3); most certificate authorities, such as Verisign and Entrust, comply with this standard.

Digital Certificate File Formats

TIBCO Enterprise Message Service supports the following file formats for digital certificates:

- PEM (Privacy Enhanced Mail)
- DER (Distinguished Encoding Rules)
- PKCS#7
- PKCS#12
- Java KeyStore (for client digital certificates)
- Entrust Store (for client digital certificates)

Private Key Formats

TIBCO Enterprise Message Service supports the following file formats for private keys:

- PEM (Privacy Enhanced Mail)
- DER (Distinguished Encoding Rules)
- PKCS#8
- PKCS#12

The EMS server uses OpenSSL to read private keys. It supports PEM, DER, PKCS8 and PKCS12 formats; it does *not* read Java KeyStore or Entrust Store files.

File Names for Certificates and Keys

For all parameters that specify the identity (digital certificate), private key, issuer (certificate chain), or trusted list of certificate authorities, valid files must be specified. Not all types of files are supported for clients and servers. The description of each parameter details which formats it supports.

Table 74 lists the valid types of files.

Table 74 File types

Extension	Description
.pem	PEM encoded certificates and keys (allows the certificate and private key to be stored together in the same file)
.der	DER encoded certificates
.p8	PKCS#8 file
.p7b	PKCS#7 file
.p12	PKCS12 file (allows the certificate and private key to be stored together in the same file)
.jks	Java KeyStore file
.epf	Entrust store file

Certificates are located in the *EMS_install_dir/certs* directory. EMS is installed with some sample certificates and private keys that are used by the sample configuration files.

The sample certificates include:

- A root, self-signed certificate and corresponding private keys in encrypted PEM and PKCS8 formats:


```
server_root.cert.pem
server_root.key.pem
server_root.key.p8
```
- A server certificate and corresponding private keys in encrypted PEM and PKCS8 formats. This certificate is issued by `server_root.cert.pem` and is used by the server:


```
server.cert.pem
server.key.pem
server.key.p8
```

- A root, self-signed certificate and corresponding private key in encrypted PEM and PKCS8 formats.
`client_root.cert.pem`
`client_root.key.pem`
`client_root.key.p8`
- A client certificate and corresponding private key in encrypted PEM and PKCS8 formats. This certificate is issued by `client_root.cert.pem` and is used by the clients:
`client.cert.pem`
`client.key.pem`
`client.key.p8`
- A PKCS12 file that includes the `client.cert.pem` client certificate, the `client.key.pem` client private key, and the `client_root.cert.pem` issuer certificate:
`client_identity.p12`

Configuring SSL in the Server

To use SSL, each instance of `tibemsd` must have a digital certificate and a private key. The server can optionally require a certificate chain or trusted certificates.

Set the server to listen for SSL connections from clients by using the `listen` parameter in `tibemsd.conf`. To specify that a port accept SSL connections, specify the SSL protocol in the `listen` parameter as follows:

```
listen = ssl://localhost:7243
```

SSL Parameters

Several SSL parameters can be set in `tibemsd.conf`. The minimum configuration is only one required parameter—`ssl_server_identity`. However, if the server's certificate file does not contain its private key, then you must specify it in `ssl_server_key`.

[SSL Server Parameters on page 224](#) provides a complete description of the SSL parameters that can be set in `tibemsd.conf`.

Command Line Options

The server accepts a few command-line options for SSL.

When starting `tibemsd`, you can specify the following options:

- `-ssl_trace`—enables tracing of loaded certificates. This prints a message to the console during startup of the server that describes each loaded certificate.
- `-ssl_debug_trace`—enables more detailed SSL tracing for debugging only; it is not for use in production systems.
- `-ssl_password`—specifies the private key password. Alternatively, you can specify this password in the `ssl_server_password` parameter in `tibemsd.conf`. If you do not supply a password using either of these methods, `tibemsd` will prompt for the password when it starts. For more information, see the description of the `ssl_password` configuration parameter.

Configuring SSL in EMS Clients

To use an SSL connection to the EMS server, a client must include these JAR files in the CLASSPATH.

- `tibcrypt.jar`
- `slf4j-api-1.4.2.jar`
- `slf4j-simple-1.4.2.jar`

These JARs are included with the TIBCO Enterprise Message Service installation, and are located in the `EMS_HOME\lib` directory.

Entrust To use Entrust with an EMS client, you must separately purchase and install the Entrust libraries. If you use the Entrust libraries, you must include them in the CLASSPATH *before* the `tibcrypt` JAR file. To use Entrust with JDK, you must download the unlimited strength policy JAR files from Sun's website and install them in your local installation of JDK. For installation and configuration details, see Entrust documentation.

Client Digital Certificates

When client authentication with a digital certificate is required by the EMS server (see the description of the `ssl_require_client_cert` parameter in [tibemsd.conf](#)), the client may combine its client certificate and private key in a single file in one of the following formats:

- PKCS#12
- Java KeyStore
- Entrust Store

You can also store the private key file separately from the client certificate file. If this is the case, the certificate and private key must be stored in one of the following formats:

- PEM
- PKCS#8

The format of the client digital certificate and private key file depends on the SSL vendor used by the client. JSSE and Entrust support different formats and combinations of formats. For more information about formats, see your SSL vendor's documentation.

Configuring SSL

A client connecting to an EMS server can configure SSL characteristics in the following ways:

- Create a connection factory that specifies the appropriate SSL parameters and use JNDI to lookup the connection factory. The server URL in the connection factory must specify the SSL protocol, and the factory must specify appropriate SSL parameters.

A preconfigured connection factory is the preferred mechanism in many situations. See [Creating Connection Factories for Secure Connections](#) and [Performing Secure Lookups](#) for details on how to create a connection factory with SSL parameters in EMS.

- Dynamically create a connection factory, as described in [Dynamically Creating Connection Factories](#) and set the global SSL parameters locally using the `TibjmsSSL` class (Java), `tibemsSSLParams` type (C), or `EMSSSL` class (C#).

Specifying any SSL parameters within a connection factory causes all global SSL parameters set with the `TibjmsSSL` class to be ignored.

Configuring a Connection Factory

You can configure a connection factory using the administration tool or the EMS Administration APIs. See [Chapter 6, Using the EMS Administration Tool](#).

When configuring a connection factory, you can specify several SSL parameters, similar to the server parameters that you can configure in [tibemsd.conf](#).



When configuring a connection factory, EMS does not verify any file names specified in the SSL parameters. At the time the factory is retrieved using JNDI, the EMS server attempts to resolve any file references. If the files do not match the supported types or the files are not found, the JNDI lookup fails with a `ConfigurationException`.



Because connection factories do not contain the `ssl_password` (for security reasons), the EMS server uses the password that is provided in the "create connection" call for user authentication. If the create connection password is different from the `ssl_password`, the connection creation will fail.

[Table 75](#) briefly describes the parameters you can set in a connection factory, and refers to additional information about each parameter. For more information about each parameter, see the description of the equivalent parameter in [tibemsd.conf on page 187](#).

Table 75 *ConnectionFactory SSL parameters (Sheet 1 of 2)*

Parameter	Description
ssl_vendor	The vendor name of the SSL implementation that the client uses.
ssl_identity	<p>The client's digital certificate.</p> <p>For more information on file types for digital certificates, see File Names for Certificates and Keys on page 469.</p>
ssl_issuer	<p>Issuer's certificate chain for the client's certificate. Supply the entire chain, including the CA root certificate. The client reads the certificates in the chain in the order they are presented in this parameter.</p> <p>Example</p> <pre>ssl_issuer = certs\CA_root.pem ssl_issuer = certs\CA_child1.pem ssl_issuer = certs\CA_child2.pem</pre> <p>For more information on file types for digital certificates, see File Names for Certificates and Keys on page 469.</p>
ssl_private_key	<p>The client's private key. If the key is included in the digital certificate in <code>ssl_identity</code>, then you may omit this parameter.</p> <p>For more information on file types for digital certificates, see File Names for Certificates and Keys on page 469.</p>
ssl_trusted	<p>List of CA certificates to trust as issuers of server certificates. Supply only CA root certificates.</p> <p>For more information on file types for digital certificates, see File Names for Certificates and Keys on page 469.</p>
ssl_verify_host	<p>Specifies whether the client should verify the server's certificate. The values for this parameter are <code>enabled</code> or <code>disabled</code>. By default, this parameter is enabled, signifying the client should verify the server's certificate.</p> <p>When <code>disabled</code>, the client establishes secure communication with the server, but does not verify the server's identity.</p>

Table 75 *ConnectionFactory SSL parameters (Sheet 2 of 2)*

Parameter	Description
<code>ssl_verify_hostname</code>	<p>Specifies whether the client should verify the name in the CN field of the server's certificate. The values for this parameter are <code>enabled</code> and <code>disabled</code>. By default, this parameter is <code>enabled</code>, signifying the client should verify the name of the connected host or the name specified in the <code>ssl_expected_hostname</code> parameter against the value in the server's certificate. If the names do not match, the client rejects the connection.</p> <p>When <code>disabled</code>, the client establishes secure communication with the server, but does not verify the server's name.</p>
<code>ssl_expected_hostname</code>	<p>The name the client expects in the CN field of the server's certificate. If this parameter is not set, the expected name is the hostname of the server.</p> <p>The value of this parameter is used when the <code>ssl_verify_hostname</code> parameter is <code>enabled</code>.</p>
<code>ssl_ciphers</code>	<p>Specifies the cipher suites that the client can use.</p> <p>Supply a colon-separated list of cipher names. Names may be either OpenSSL names, or longer descriptive names.</p> <p>For more information, see Specifying Cipher Suites on page 476.</p>
<code>ssl_auth_only</code>	<p>Specifies whether SSL should be used to encrypt all server-client communications, or only client authentication.</p> <p>When <code>enabled</code>, the client requests SSL be used only for authentication. The server then uses TCP communications for further data exchange. When <code>disabled</code> or absent, all communication between the client and server must be SSL encrypted.</p> <p>For an overview of this feature, see SSL Authentication Only on page 482.</p>
<code>ssl_rand_egd</code>	<p>The path for the entropy gathering daemon (EGD), if one is installed. This daemon generates random data for the client.</p>

Specifying Cipher Suites

On the EMS server, specify cipher suites using the `ssl_server_ciphers` configuration parameter in `tibemsd.conf`. For more information about server configuration files, see [Chapter 7, Using the Configuration Files, on page 185](#).

For clients connecting with a connection factory, specify cipher suites using the `ssl_ciphers` connection factory parameter. For more information, see [Configuring SSL in EMS Clients on page 472](#).

Syntax for Cipher Suites

EMS uses OpenSSL for SSL support. Therefore, the cipher suite names can be specified as the OpenSSL name for the cipher suite.

When specifying cipher suites, the usual way to specify more than one cipher suite is to separate each suite name with a colon (:) character. Alternatively, you can use spaces and commas to separate names.

Java Client Syntax

The syntax for specifying the list of cipher suites is different for Java clients than for any other location where cipher suites can be specified. For Java clients, you specify a qualifier (for example, + to add the suite) followed by the cipher suite name. Cipher suite names are case-sensitive. [Table 76](#) describes the qualifiers you can use when specifying cipher suite names in a `ConnectionFactory` for Java clients.

Table 76 Qualifiers for Cipher Suites in Java Clients

Qualifier	Description
+	Add the cipher to the list of ciphers.
-	Remove the cipher from the list of ciphers.
>	Move the cipher to the end of the list.
<	Move the cipher to the beginning of the list.
ALL	All ciphers from the list (except null ciphers). You can use this keyword to add or remove all ciphers. At least one cipher suite must be present, otherwise the SSL connection fails to initialize. So, if you use -ALL, you must subsequently add the desired ciphers to the list.

This example specifies cipher suites in the `ssl_ciphers` connection factory parameter in a Java client:

```
-ALL : +RC4-MD5 : +DES-CBC-SHA : <DES-CBC3-SHA
```

This example specifies cipher suites using full names:

```
-ALL : +SSL_RSA_WITH_RC4_128_MD5 : +SSL_RSA_WITH_DES_CBC_SHA : <SSL_RSA_WITH_3DES_EDE_CBC_SHA
```

Syntax for All Other Cipher Suite Specifications

For any cipher suite list that is not specified in a connection factory of a Java client, use the OpenSSL syntax. In particular, C clients and the `ssl_server_ciphers` configuration parameter require OpenSSL syntax.

In OpenSSL syntax, specifying a cipher suite name adds that cipher suite to the list. Each cipher suite name can be preceded by a qualifier. Cipher suite names are case-sensitive. [Table 77](#) describes the qualifiers available using OpenSSL syntax.

Table 77 *OpenSSL Qualifiers for Cipher Suites (Sheet 1 of 2)*

Qualifier	Description
/	<p>When entered as the first item in the list, this option causes EMS to begin with an empty list, and add the ciphers that follow the slash.</p> <p>If the / does not prefix the cipher list, then EMS prefixes the cipher list with the OpenSSL cipher string <code>DEFAULT</code>.</p> <p>This modifier can only be used at the beginning of the list. If the / appears elsewhere, the syntax of the cipher suite list will be incorrect and cause an error.</p>
+	<p>Moves the cipher to the end of the list.</p> <p>This qualifier is used to move an existing cipher. It can not be used to add a new cipher to the list.</p>
-	<p>Remove the cipher from the list of ciphers. When this option is used, the cipher can be added later on in the list of ciphers.</p>
!	<p>Permanently disable the cipher within the list of ciphers. Use this option if you wish to remove a cipher and you do not want later items in the list to add the cipher to the list. This qualifier takes precedence over all other qualifiers.</p>

Table 77 *OpenSSL Qualifiers for Cipher Suites (Sheet 2 of 2)*

Qualifier	Description
ALL	All ciphers from the list (except null ciphers). You can use this keyword to add or remove all ciphers. At least one cipher suite must be present or the SSL connection fails to initialize. So, after using -ALL, you should add at least one cipher to the list.
@STRENGTH	Sort the cipher list by key length.

This example specifies cipher suites in the `ssl_server_ciphers` configuration parameter.

```
ssl_server_ciphers = -ALL:RC4-MD5:DES-CBC-SHA:DES-CBC3-SHA
```

This example illustrates disables RC4-MD5, then adds all other ciphers:

```
ssl_server_ciphers = !RC4-MD5:ALL
```

Default Cipher List The EMS server and C client library hard-code a default cipher list, which is equivalent to `ALL: !ADH: RC4+RSA: +SSLv2: @STRENGTH`.

Supported Cipher Suites

In general, the EMS server and C client library support all cipher suites that OpenSSL supports, except IDEA, RC-5 and CAST. For a complete list, see current OpenSSL documentation.

Supported Cipher Suites for Java Clients

Java clients support *only* the cipher suites listed in [Table 78](#). For convenience, the table lists both the standard name and the OpenSSL name for each cipher suite.

Table 78 *Supported Cipher Suites in Java API (Sheet 1 of 4)*

Suite Name (OpenSSL Name)	Export	Key Exch	Auth	Encrypt	Key Size	MAC
SSL_RSA_WITH_RC4_128_MD5 (RC4-MD5)		RSA	RSA	RC4	128	MD5

Table 78 Supported Cipher Suites in Java API (Sheet 2 of 4)

Suite Name (OpenSSL Name)	Export	Key Exch	Auth	Encrypt	Key Size	MAC
SSL_RSA_WITH_RC4_128_SHA (RC4-SHA)						
		RSA	RSA	RC4	128	SHA1
SSL_RSA_WITH_DES_CBC_SHA (DES-CBC-SHA)						
		RSA	RSA	DES	56	SHA1
SSL_RSA_WITH_3DES_EDE_CBC_SHA (DES-CBC3-SHA)						
		RSA	RSA	3-DES	168	SHA1
SSL_RSA_EXPORT_WITH_RC4_40_MD5 (EXP-RC4-MD5)						
	Yes	RSA(512)	RSA	RC4	40	MD5
SSL_RSA_EXPORT_WITH_DES_40_CBC_SHA (EXP-DES-CBC-SHA)						
	Yes	RSA(512)	RSA	DES	40	SHA1
SSL_DHE_DSS_WITH_3DES_EDE_CBC_SHA (EDH-DSS-DES-CBC3-SHA)						
		DH	DSS	3-DES	168	SHA1
SSL_DHE_RSA_WITH_3DES_EDE_CBC_SHA (EDH-RSA-DES-CBC3-SHA)						
		DH	RSA	3-DES	168	SHA1
SSL_DHE_DSS_WITH_DES_CBC_SHA (EDH-DSS-DES-CBC-SHA)						
		DH	DSS	DES	56	SHA1

Table 78 Supported Cipher Suites in Java API (Sheet 3 of 4)

Suite Name (OpenSSL Name)	Export	Key Exch	Auth	Encrypt	Key Size	MAC
SSL_DHE_RSA_WITH_DES_CBC_SHA (EDH-RSA-DES-CBC-SHA)		DH	RSA	DES	56	SHA1
SSL_DHE_DSS_EXPORT_WITH_DES_40_CBC_SHA (EXP-EDH-DSS-DES-CBC-SHA)	Yes	DH(512)	DSS	DES	40	SHA1
SSL_DHE_RSA_EXPORT_WITH_DES_40_CBC_SHA (EXP-EDH-RSA-DES-CBC-SHA)	Yes	DH(512)	RSA	DES	40	SHA1
TLS_RSA_WITH_AES_128_CBC_SHA (AES128-SHA)		RSA	RSA	AES	128	SHA1
TLS_RSA_WITH_AES_256_CBC_SHA (AES256-SHA)		RSA	RSA	AES	256	SHA1
TLS_DHE_DSS_WITH_AES_128_CBC_SHA (DHE-DSS-AES128-SHA)		DH	DSS	AES	128	SHA1
TLS_DHE_DSS_WITH_AES_256_CBC_SHA (DHE-DSS-AES256-SHA)		DH	DSS	AES	256	SHA1
TLS_DHE_RSA_WITH_AES_128_CBC_SHA (DHE-RSA-AES128-SHA)		DH	RSA	AES	128	SHA1

Table 78 Supported Cipher Suites in Java API (Sheet 4 of 4)

Suite Name (OpenSSL Name)	Export	Key Exch	Auth	Encrypt	Key Size	MAC
TLS_DHE_RSA_WITH_AES_256_CBC_SHA (DHE-RSA-AES256-SHA)						
		DH	RSA	AES	256	SHA1



Enterprise Message Service does not support these cipher suites:

- SSL_RSA_WITH_NULL_SHA
- SSL_RSA_WITH_NULL_MD5

Although they are not supported, they are included in the interface definition only to allow old programs to compile correctly. Use the SSL authentication only feature in place of these cipher suites. See [SSL Authentication Only](#) below for more information.

Supported Cipher Suites for .NET Clients

.NET client support only the following cipher suites:

- RC4-MD5
- RC4-SHA
- DES-CBC3-SHA
- DES-CBC-SHA
- EXP-RC2-CBC-MD5
- EDH-DSS-DES-CBC3-SHA
- EDH-DSS-DES-SHA
- EXP-RC4-MD5
- AES128-SHA



Some newer Windows platforms, such as Windows Server 2008 and Windows 7, don't support weaker ciphers (like EXP-RC4-MD5). These platforms support the stronger ciphers (like AES128-SHA). Refer to your MSDN documentation or contact Microsoft support for complete details on supported ciphers on specific Windows platforms.

SSL Authentication Only

EMS servers can use SSL for secure data exchange (standard usage), or only for client authentication. This section describes the use of SSL for client authentication.

Motivation Some applications require strong or encrypted authentication, but do not require message encryption.

In this situation, application architects could configure SSL with a null cipher. However, this solution incurs internal overhead costs of SSL calls, decreasing message speed and throughput.

For optimal performance, the preferred solution is to use SSL only to authenticate clients, and then avoid SSL calls thereafter, using ordinary TCP communications for subsequent data exchange. Message performance remains unaffected.

Preconditions All three of these preconditions must be satisfied to use SSL only for authentication:

- The server and clients must both be release 4.2 or later. (If not, EMS behavior reverts to using SSL for all communications throughout the life of the connection.)
- The server must explicitly enable the parameter `ssl_auth_only` in the `tibemsd.conf` configuration file.
- The client program must request a connection that uses SSL for authentication only. Clients can specify this request in factories by enabling the `ssl_auth_only` parameter, or by calling:

- Java: `TibjmsSSL.setAuthOnly`
- C: `tibemsSSLParams_SetAuthOnly`
- C#: `EMSSSL.SetAuthOnly`

See Also Server parameter [ssl_auth_only on page 228](#)
 Client parameter [ssl_auth_only on page 475](#)

Enabling FIPS Compliance

You can enable TIBCO Enterprise Message Service to run in compliance with Federal Information Processing Standard (FIPS), Publication 140-2.

Enabling the EMS Server



The EMS server supports FIPS compliance only on Windows, Linux, and Solaris 10 (x86) platforms. On UNIX, only `tibemsd64`, the 64-bit version of the server, is supported. No 32-bit support is provided.

To enable FIPS 140-2 operations in the EMS server:

- Set the `fips140-2` parameter in the main configuration file to `true`.
- Ensure that incompatible parameters, listed below, are not included in the server configuration files.

When `fips140-2` is enabled, on start-up the EMS server initializes in compliance with FIPS 140-2. If the initialization is successful, the EMS server prints a message indicating that it is operating in this mode. If the initialization fails, the server exits (regardless of the `startup_abort_list` setting).

Incompatible Parameters

In order to operate in FIPS compliant mode, you must not include these parameters in the `tibemsd.conf` file:

- `ssl_dh_size`
- `ssl_server_ciphers`
- `ldap_tls_rand_file`
- `ldap_tls_cipher_suite`
- `ft_ssl_ciphers`

These parameters cannot be included in the `routes.conf` file:

- `ssl_ciphers`

Enabling EMS Clients

Java and C client applications can operate in FIPS compliance:

- **Java Clients** Java clients that use the Entrust implementation of SSL, rather than the JSSE that is included with EMS, can operate in FIPS 140-2 complaint mode.

To enable FIPS 140-2 operations in the Java client:

- Set the `com.tibco.security.FIPS` property to `true` before calling any EMS methods.
- Download and install the Java Cryptography Extension (JCE) Unlimited Strength Jurisdiction Policy Files for your JDK installation. These files are available on the Sun Microsystems website.

For more information about using Entrust, see [Configuring SSL in EMS Clients on page 472](#).

- **C Clients** C clients that link to the dynamic EMS libraries can operate in FIPS 140-2 compliant mode. FIPS compliance is not available with static libraries.

To enable FIPS 140-2 operations in the C client, use compliant OpenSSL libraries, and initialize the libraries to enable FIPS 140-2 operations before calling any EMS functions.



C libraries support FIPS compliance only on Windows, Linux, and Solaris 10 (x86) platforms. On UNIX, only the 64-bit C libraries are supported. No 32-bit support is provided.

Chapter 19 **Fault Tolerance**

This chapter describes the fault tolerance features of TIBCO Enterprise Message Service.

Topics

- [Fault Tolerance Overview, page 486](#)
- [Shared State Failover Process, page 488](#)
- [Unshared State Failover Process, page 492](#)
- [Shared State, page 495](#)
- [Configuring Fault-Tolerant Servers, page 499](#)
- [Configuring Fault Tolerance in Central Administration, page 502](#)
- [Configuring Clients for Fault-Tolerant Connections, page 504](#)
- [Configuring Clients for Unshared State Connections, page 507](#)

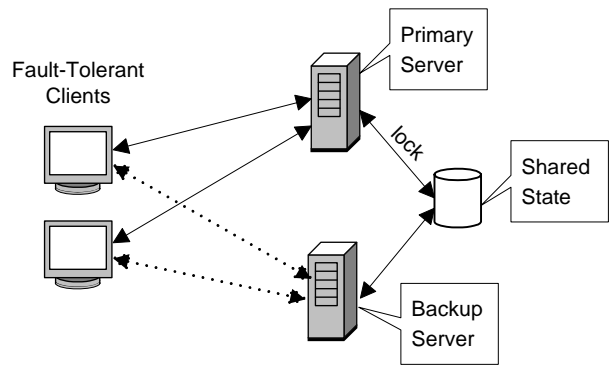
Fault Tolerance Overview

You can arrange TIBCO Enterprise Message Service servers for fault-tolerant operation by configuring a pair of servers—one primary and one backup. The primary server accepts client connections, and interacts with clients to deliver messages. If the primary server fails, the backup server resumes operation in its place. (We do not support more than two servers in a fault-tolerant configuration.)

Shared State

A pair of fault-tolerant servers can have access to shared state, which consists of information about client connections and persistent messages. This information enables the backup server to properly assume responsibility for those connections and messages. [Figure 23](#) illustrates a fault-tolerant configuration of EMS.

Figure 23 Primary and Backup Servers with Shared State



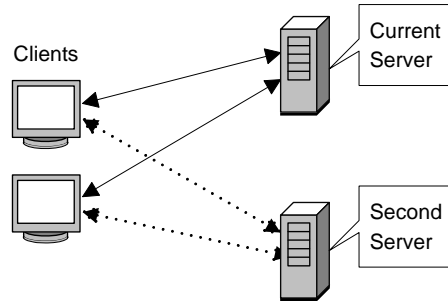
Locking To prevent the backup server from assuming the role of the primary server, the primary server locks the shared state during normal operation. If the primary server fails, the lock is released, and the backup server can obtain the lock.

Unshared State Failover

You can also include backup servers that do not share state. As with shared state, a second server assumes responsibility for connections and messages after the failure of the current server. However, unlike shared state, unshared state is controlled by the EMS client, and unshared state failover is not as fault-tolerant as shared state failover. Because the state is not shared among servers, messages can be lost, duplicated, or delivered out-of-order across the failover process.

[Figure 24](#) illustrates an unshared state fault-tolerant configuration of EMS.

Figure 24 Current and Second Servers with Unshared State



Configuration Files

When a primary server fails, its backup server assumes the status of the primary server and resumes operation. Before becoming the new primary server, the backup server re-reads all of its configuration files. If the two servers share configuration files, then administrative changes to the old primary carry over to the new primary.



When fault-tolerant servers share configuration files, you *must* limit configuration changes to the current primary server only. Separately reconfiguring the backup server can cause it to overwrite the shared configuration files; unintended misconfiguration can result.

Shared State Failover Process

This section presents details of the shared state failover sequence.

Detection

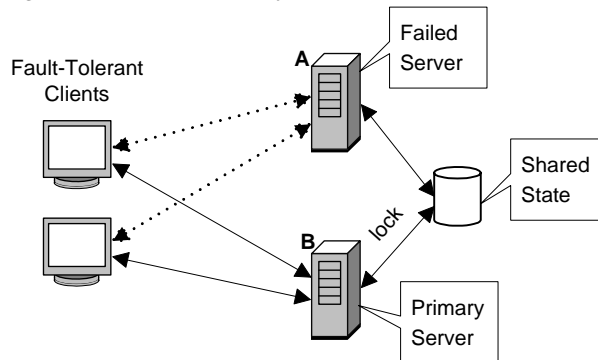
A backup server detects a failure of the primary in either of two ways:

- *Heartbeat Failure*—The primary server sends heartbeat messages to the backup server to indicate that it is still operating. When a network failure stops the servers from communicating with each other, the backup server detects the interruption in the steady stream of heartbeats. For details, see [Heartbeat Parameters on page 491](#).
- *Connection Failure*—The backup server can detect the failure of its TCP connection with the primary server. When the primary process terminates unexpectedly, the backup server detects the broken connection.

Response

When a backup server (B) detects the failure of the primary server (A), then B attempts to assume the role of primary server. First, B obtains the lock on the current shared state. When B can access this information, it becomes the new primary server.

Figure 25 Failed Primary Server



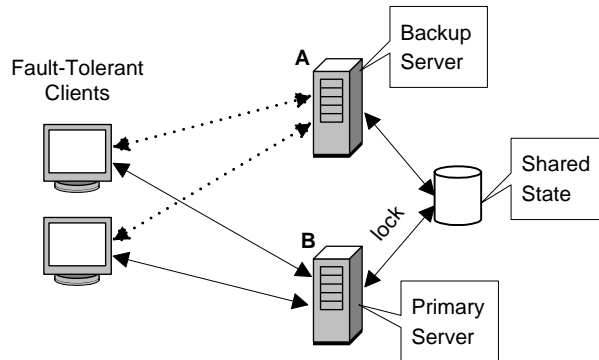
Lock Unavailable

If B cannot obtain the lock immediately, it alternates between attempting to obtain the lock (and become the primary server), and attempting to reconnect to A (and resume as a backup server)—until one of these attempts succeeds.

Role Reversal

When B becomes the new primary server, A can restart as a backup server, so that the two servers exchange roles.

Figure 26 Recovered Server Becomes Backup



Client Transfer

Clients of A that are configured to failover to backup server B automatically transfer to B when it becomes the new primary server. B reads the client's current state from the shared storage to deliver any persistent messages to the client.

Client Notification

Client applications can receive notification when shared state failover occurs.

Java

To receive notification, Java client programs set the system property `tibco.tibjms.ft.switch.exception` to any value, and define an `ExceptionListener` to handle failover notification; see the class `com.tibco.tibjms.Tibjms` in *TIBCO Enterprise Message Service Java API Reference*.

C

To receive notification, C client programs call `tibems_setExceptionOnFTSwitch(TIBEMS_TRUE)` and register the exception callback in order to receive the notification that the reconnection was successful.

C#

To receive notification, .NET client programs call `Tibems.SetExceptionOnFTSwitch(true)`, and define an exception listener to handle failover notification; see the method `Tibems.SetExceptionOnFTSwitch` on page 294 in *TIBCO Enterprise Message Service .NET API Reference*.

Message Redelivery

Persistent	When a failure occurs, messages with delivery mode <code>PERSISTENT</code> , that were not successfully acknowledged before the failure, are redelivered.
Synchronous Mode	When using durable subscribers, EMS guarantees that a message with <code>PERSISTENT</code> delivery mode and written to a <code>store</code> with the property <code>mode=sync</code> , will not be lost during a failure.
Delivery Succeeded	Any messages that have been successfully acknowledged or committed are not redelivered, in compliance with the JMS specification.
Topics	All topic subscribers continue normal operation after a failover.

Transactions

A transaction is considered *active* when at least one message has been sent or received by the session, and the transaction has not been successfully committed.

After a failover, attempting to commit the active transaction results in a `javax.jms.TransactionRolledBackException`. Clients that use transactions must handle this exception, and resend any messages sent during the transaction. The backup server automatically redelivers any messages that were delivered to the session during the transaction that rolled back.

Queues

For queue receivers, any messages that have been sent to receivers, but have not been acknowledged before the failover, may be sent to other receivers immediately after the failover.

A receiver trying to acknowledge a message after a failover may receive the `javax.jms.IllegalStateException`. This exception signifies that the attempted acknowledgement is for a message that has already been sent to another queue receiver. This exception only occurs in this scenario, or when the session or connection have been closed. This exception cannot occur if there is only one receiver at the time of a failover, but it may occur for exclusive queues if more than one receiver was started for that queue.

When a queue receiver catches a `javax.jms.IllegalStateException`, the best course of action is to call the `Session.recover()` method. Your application program should also be prepared to handle redelivery of messages in this situation. All queue messages that can be redelivered to another queue receiver after a failover always have the header field `JMSRedelivered` set to `true`; application programs must check this header to avoid duplicate processing of the same message in the case of redelivery.



Acknowledged messages are never redelivered (in compliance with the JMS specification). The case described above occurs when the application cannot acknowledge a message because of a failover.

Heartbeat Parameters

When the primary server heartbeat stops, the backup server waits for its activation interval (elapsed time since it detected the most recent heartbeat); then the backup server retrieves information from shared storage and assumes the role of primary server.

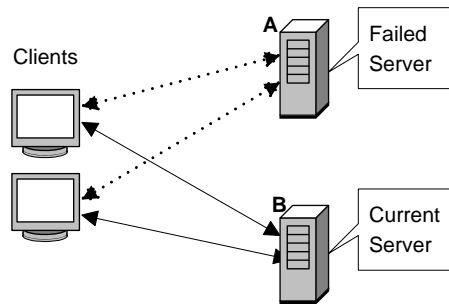
The default heartbeat interval is 3 seconds, and the default activation interval is 10 seconds. The activation interval must be at least twice the heartbeat interval. Both intervals are specified in seconds. You can set these intervals in the server configuration files. See [Fault Tolerance Parameters on page 213](#) for details.

Unshared State Failover Process

This section presents details of the unshared state failover sequence. Detailed configuration information is provided in [Configuring Clients for Unshared State Connections on page 507](#).

Detection Unshared state failover is initiated by the EMS client. When a client setup for unshared state detects a lost connection to server (A), it attempts to connect to server (B), as defined in the connection factory.

Figure 27 Unshared State Failover



Response Clients with unshared state connections automatically connect to B after losing the connection to A.

When clients setup for unshared state detect lost connections to server A, they create new connections to server B. All runtime objects from the client's connection are recreated, including sessions, destinations, message producers, and message consumers.

Because unshared state is defined in the connection factory, B remains the current server as long as the connection is active. If the connection to B is lost, clients attempt to connect to another server defined in the connection factory.

Message Loss Because B does not have access to persistent messages that were not delivered or acknowledged prior to the failover, some messages may be lost or delivered out of order across the failover process. To prevent message loss, use shared state failover.

Unsupported Features These features and Java classes are not supported with unshared state connections:

- XA transactions
- Multicast

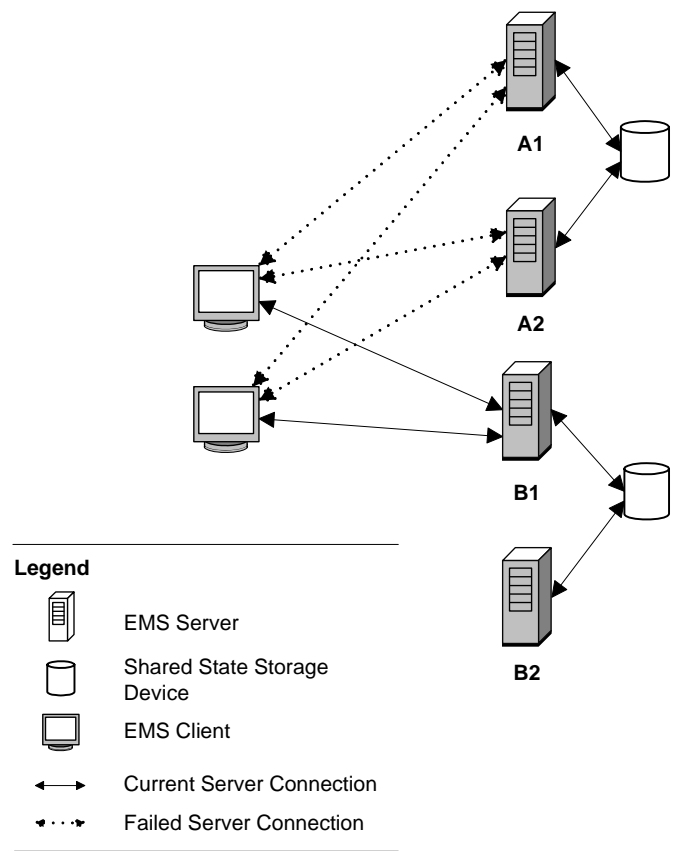
- Durable topic subscribers
- ConnectionConsumer
- ServerSession
- ServerSessionPool
- QueueRequestor
- TopicRequestor

Dual State Failover

An unshared state connection factory can include shared-state server pairs in its list of backup servers. When both shared state and unshared state servers are included, the failover process is a combination of both types of failover.

[Figure 28](#) illustrates the dual state failover process.

Figure 28 Dual State Failover Process



In this example, servers A1 and A2 share state. Servers B1 and B2 also share state. However, A1 and A2 do *not* share state with B1 and B2.

The EMS clients created connections using unshared state connection factories. The initial server connections were with server A1. When the connection to A1 failed, the failover process proceeded as described in [Shared State Failover Process on page 488](#), and the clients connect to A2.

A2 then failed, before A1 restarted. The clients next created connections to B1, recreating all runtime objects from the connection (as described above in [Unshared State Failover Process](#)). B1 is now the current server. Because B1 and B2 share state, If B1 fails, B2 becomes the current server.

Shared State

For the most robust failover protection, the primary server and backup server must share the same state. Server state includes three categories of information:

- persistent message data (for queues and topics)
- client connections of the primary server
- metadata about message delivery

During a failover, the backup server re-reads all shared state information.

Implementing Shared State

We recommend that you implement shared state using shared storage devices. The shared state must be accessible to both the primary and backup servers.

Support Criteria

Several options are available for implementing shared storage using a combination of hardware and software. EMS requires that your storage solution guarantees all four criteria in [Table 79](#).



Always consult your shared storage vendor *and* your operating system vendor to ascertain that the storage solution you select satisfies all four criteria.

Table 79 Shared Storage Criteria for Fault Tolerance

Criterion	Description
Write Order	The storage solution must write data blocks to shared storage in the same order as they occur in the data buffer. (Solutions that write data blocks in any other order (for example, to enhance disk efficiency) do <i>not</i> satisfy this requirement.)
Synchronous Write Persistence	Upon return from a synchronous write call, the storage solution guarantees that all the data have been written to durable, persistent storage.

Table 79 Shared Storage Criteria for Fault Tolerance

Criterion	Description
Distributed File Locking	<p>The EMS servers must be able to request and obtain an exclusive lock on the shared storage. The storage solution must <i>not</i> assign the locks to two servers simultaneously. (See Software Options on page 497.)</p> <p>EMS servers use this lock to determine the primary server.</p>
Unique Write Ownership	<p>The EMS server process that has the file lock must be the only server process that can write to the file. Once the system transfers the lock to another server, pending writes queued by the previous owner must fail.</p>

Hardware Options

Consider these examples of commonly-sold hardware options for shared storage:

- Dual-Port SCSI device
- Storage Area Network (SAN)
- Network Attached Storage (NAS)

SCSI and SAN Dual-port SCSI and SAN solutions generally satisfy the [Write Order](#) and [Synchronous Write Persistence](#) criteria. (The clustering software must satisfy the remaining two criteria.) As always, you must confirm all four requirements with your vendors.

NAS NAS solutions require a CS (rather than a CFS) to satisfy the [Distributed File Locking](#) criterion (see below).

Some NAS solutions satisfy the criteria, and some do not; you must confirm all four requirements with your vendors.

NAS with NFS When NAS hardware uses NFS as its file system, it is particularly difficult to determine whether the solution meets the criteria. Our research indicates the following conclusions:

- NFS v2 and NFS v3 definitely do *not* satisfy the criteria.
- NFS v4 with TCP *might* satisfy the criteria. Consult with the NAS vendor to verify that the NFS server (in the NAS) satisfies the criteria. Consult with the operating system vendor to verify that the NFS client (in the OS on the server host computer) satisfies the criteria. When both vendors certify that their components cooperate to guarantee the criteria, then the shared storage solution supports EMS.

For more information on how the EMS locks shared store files, see [How EMS Manages Access to Shared Store Files on page 120](#).

Software Options

Consider these examples of commonly-sold software options:

- **Cluster Server (CS)**
A cluster server monitors the EMS server processes and their host computers, and ensures that exactly one server process is running at all times. If the primary server fails, the CS restarts it; if it fails to restart the primary, it starts the backup server instead.
- **Clustered File System (CFS)**
A clustered file system lets the two EMS server processes run simultaneously. It even lets both servers mount the shared file system simultaneously. However, the CFS assigns the lock to only one server process at a time. The CFS also manages operating system caching of file data, so the backup server has an up-to-date view of the file system (instead of a stale cache).

With dual-port SCSI or SAN hardware, either a CS or a CFS might satisfy the [Distributed File Locking](#) criterion. With NAS hardware, only a CS can satisfy this criterion (CFS software generally does not). Of course, you must confirm all four requirements with your vendors.

Messages Stored in Shared State

Messages with `PERSISTENT` delivery mode are stored, and are available in the event of primary server failure. Messages with `NON_PERSISTENT` delivery mode are not available if the primary server fails.

For more information about recovery of messages during failover, see [Message Redelivery on page 490](#).

Storage Files

By default, the `tibemsd` server creates three file-based stores to store shared state:

- `$sys.failsafe`—This store holds persistent messages using synchronous I/O calls.
- `$sys.nonfailsafe`—This file stores messages using asynchronous I/O calls.
- `$sys.meta`—This store holds state information about durable subscribers, fault-tolerant connections, and other metadata.

These stores are fully customizable through parameters in the stores configuration file. More information about these files and the default configuration settings are fully described in [stores.conf on page 253](#).

To prevent two servers from using the same store file, each server restricts access to its store file for the duration of the server process. For more information on how the EMS manages shared store files, see [How EMS Manages Access to Shared Store Files on page 120](#).



These default files can be changed or modified. See [Default Store Files on page 31](#) for more information.

Storage Parameters

Several configuration parameters apply to EMS storage files (even when fault-tolerant operation is not configured); see [Storage File Parameters on page 206](#).

Configuring Fault-Tolerant Servers

Shared State

To configure an EMS server as a fault-tolerant backup, set these parameters in its main configuration file (or on the server command line):

- `server` Set this parameter to the same server name in the configuration files of both the primary server and the backup server.
- `ft_active` In the configuration file of the primary server, set this parameter to the URL of the backup server. In the configuration file of the backup server, set this parameter to the URL of the primary server.

When the backup server starts, it attempts to connect to the primary server. If it establishes a connection to the primary, then the backup server enters standby mode. If it cannot establish a connection to the primary, then the backup server assumes the role of the primary server (in active mode).

While the backup server is in standby mode, it does not accept connections from clients. To administer the backup server, the `admin` user can connect to it using the administration tool.

Authorization and Fault-Tolerant Servers

EMS authorization interacts with fault tolerance. If `authorization` is enabled and the two EMS Servers are configured for fault tolerance, then both servers in a fault-tolerant pair must be configured as follows:

- The `tibemsd.conf` file for each server must have the same server name and password (the `server` and `password` parameters must be the same on each server).
- The user name and password in the `users.conf` file for each server must match the values of the `server` and `password` parameters in the `tibemsd.conf` file.



If the two EMS Servers are not sharing a `users.conf` file, make sure that you create a user with the same name as the EMS Server, and set the user's password with the value of the "server" password.

For example, you have two EMS Servers (Server 1 and Server 2) that are named "EMS-SERVER" and are to use a password of "mySecret", but which do not share a `users.conf` file. To set the user names and passwords, start the EMS Administration Tool on each server, as described in [Using the EMS Administration Tool on page 123](#), and do the following.

From the active (Server 1), enter:

```
set server password=mySecret
create user EMS-SERVER password=mySecret
```

From the backup (Server 2), enter:

```
set server password=mySecret
create user EMS-SERVER password=mySecret
```

From the active (Server 1), enter:

```
set server authorization=enabled
```

From the backup (Server 2), enter:

```
set server authorization=enabled
```

SSL

You can use SSL to secure communication between a pair of fault-tolerant servers.

Parameters in the main configuration file (`tibemsd.conf`) affect this behavior. The relevant parameters all begin with the prefix `ft_ssl`.

The server initializing a secure connection to another server uses the `ft_ssl` parameters to determine the properties of its secure connection to the other server. The receiving server validates the incoming connection against its own `ssl_` parameters. For more information about `ft_ssl` parameters, see [Fault Tolerance Parameters on page 213](#). For more information about `ssl_` parameters, see [SSL Server Parameters on page 224](#).

See Also [Chapter 18, Using the SSL Protocol, on page 465](#)

Reconnect Timeout

When a backup server assumes the role of the primary server during failover, clients attempt to reconnect to the backup server (that is, the new primary) and continue processing their current message state. Before accepting reconnects from the clients, the backup server reads its message state from the shared state files.

You can instruct the server to clean up state information for clients that do not reconnect before the time limit specified by the `ft_reconnect_timeout` configuration parameter. The `ft_reconnect_timeout` time starts once the server has fully recovered the shared state, so this value does not account for the time it takes to recover the store files. See [ft_reconnect_timeout on page 214](#) for details.

Unshared State

When configuring a fault tolerant pair that does not share state, you must ensure that both servers use identical configurations. This is especially important for these configuration settings:

- **Destinations** Both servers must support the same destinations.
- **Routes** Messages must be able to arrive at the endpoints, using equivalent or identical routes across servers.
- **Access Control** Access control must be setup identically in both servers, so that the `users.conf`, `groups.conf`, and `acl.conf` file settings match.
- **SSL** When SSL is deployed, both servers must use the same certificate(s).

Configuring Fault Tolerance in Central Administration

Central Administration uses the same JSON configuration file to manage both servers in a fault tolerant pair. Primary and secondary server roles are determined when the servers are started.

All but two configuration settings are shared by both EMS servers: the `listen` and `ft_active` parameters are configured separately.

- The primary server listens for client connection on ports defined in the main Server Properties page, in the Primary Listens section. After a failover, it listens for the secondary server on the Secondary Listens URL that is flagged using the FT Active radio button on the Fault Tolerance properties page.
- The secondary server listens for the primary server using the Primary Listens URL that is flagged with the FT Active radio button on the main Server Properties page. If the secondary server becomes active, it listens for client connections using the Secondary Listen URLs defined on the Fault Tolerance page.

For more information on Central Administration, see *TIBCO Enterprise Message Service Central Administration*.

- Procedure To configure a fault tolerant server pair using Central Administration:
1. Configure the primary server as usual.
 2. On the Server Properties page, designate the URL on which the secondary server listens for the primary server by clicking the FT Active radio button next to the desired Listens URL.
 3. On the Fault Tolerance properties page, configure the Secondary Listens URLs that the backup server uses to listen for client connections in the event that it becomes the primary server.
 4. Designate the URL on which the primary server listens for the secondary server, should a failure occur and the secondary server becomes active. Click the FT Active radio button next to the desired Secondary Listens URL.
 5. Configure the remaining fault tolerance properties on the Fault Tolerance page.
 6. Deploy the configuration changes.
 7. Start the primary and secondary EMS servers using the method described in [Starting Fault Tolerant Server Pairs on page 109](#).

Configuration Errors When an EMS server is started, the fault tolerance mechanism is triggered by the presence of a URL in the Secondary Listens list of a primary `tibemsd`, or by that of a URL in the Primary Listens list of a secondary `tibemsd`.

Once fault tolerance is triggered, the EMS server generates an error if it finds that the "FT Active" switch was not assigned to any URL in its peer's list. If `CONFIG_ERRORS` is present in the `startup_abort_list` parameter, the `tibemsd` aborts startup. Otherwise, the `tibemsd` cancels fault tolerance and starts without checking its peer. This results in a file lock error for the EMS server that is started second.

Configuring Clients for Fault-Tolerant Connections

When a backup server assumes the role of the primary server during failover, clients attempt to reconnect to the backup server (that is, the new primary). To enable a client to reconnect, you must specify the URLs of both servers when creating a connection.

Specify multiple servers as a comma-separated list of URLs. Both URLs must use the same protocol (either `tcp` or `ssl`). For example, to identify the first server as `tcp://server0:7222`, and the second server as `tcp://server1:7344` (if first server is not available), you can specify:

```
serverUrl=tcp://server0:7222, tcp://server1:7344
```

The client attempts to connect to each URL in the order listed. If a connection to one URL fails, the client tries the next URL in the list. The client tries the URLs in sequence until all URLs have been tried. If the first failed connection was not the first URL in the list, the attempts wrap to the start of the list (so each URL is tried). If none of the attempts succeed, the connection fails.

For information on how to lookup a fault-tolerance URL in the EMS naming service, see [Performing Fault-Tolerant Lookups on page 366](#).



The reconnection logic in the client is triggered by the specifying multiple URLs when connecting to a server. If no backup server is present, the client must still provide at least two URLs (typically pointing to the same server) in order for it to automatically reconnect to the server when it becomes available after a failure.



When messages are sent in non-persistent or reliable modes, the consumer does not normally wait for a server reply to its acknowledgements. However, a fault tolerant consumer does wait for a server reply (when using an session mode other than `DUPS_OK_ACKNOWLEDGE` or `EXPLICIT_CLIENT_DUPS_OK_ACKNOWLEDGE`). This is true for shared state configurations. Unshared state configurations, which tolerate lost, duplicated, and out-of-order messages during a failover, do not wait for server acknowledgements.

Specifying More Than Two URLs

Even though there are only two servers (the primary and backup servers), clients can specify more than two URLs for the connection. For example, if each server has more than one listen address, a client can reconnect to the same server at a different address (that is, at a different network interface).

Setting Reconnection Failure Parameters

EMS allows you to establish separate parameters for initial connection attempts and reconnection attempts. How to set the initial connection attempt parameters is described in [Setting Connection Attempts, Timeout and Delay Parameters on page 333](#). This section describes the parameters you can establish for reconnection attempts following a fault-tolerant switchover.

The reason for having separate connect and reconnect attempt parameters is that there is a limit imposed by the operating system to the number of connection attempts the EMS server can handle at any particular time. (For example, in Unix, this limit is adjusted by the `ulimit` setting.) Under normal circumstances, each connect attempt is distributed so it is less likely for the server to exceed its maximum accept queue. However, during a fault-tolerant switchover, all of the clients automatically try to reconnect to the backup server at approximately the same time. When the number of connections is large, it may require more time for each client to reconnect than for the initial connect.

By default, a client will attempt reconnection 4 times with a 500 ms delay between each attempt. You can modify these settings in the `factories.conf` file or by means of your client connection factory API, as demonstrated by the examples in this section.

The following examples establish a reconnection count of 10, a delay of 1000 ms and a timeout of 1000 ms.

Java

Use the `TibjmsConnectionFactory` object's `setReconnAttemptCount()`, `setReconnAttemptDelay()`, and `setReconnAttemptTimeout()` methods to establish new reconnection failure parameters:

```
factory.setReconnAttemptCount(10);
factory.setReconnAttemptDelay(1000);
factory.setReconnAttemptTimeout(1000);
```

C

Use the `tibemsConnectionFactory_SetReconnectAttemptCount`, `tibemsConnectionFactory_SetReconnectAttemptDelay`, and `tibemsConnectionFactory_SetReconnectAttemptTimeout` functions to establish new reconnection failure parameters:

```
status = tibemsConnectionFactory_SetReconnectAttemptCount(
    factory, 10);

status = tibemsConnectionFactory_SetReconnectAttemptDelay(
    factory, 1000);

status = tibemsConnectionFactory_SetReconnectAttemptTimeout(
```

```
factory, 1000);
```

C#

Use the `ConnectionFactory.SetReconnAttemptCount`, `ConnectionFactory.SetReconnAttemptDelay`, and `ConnectionFactory.SetReconnAttemptTimeout` methods to establish new reconnection failure parameters:

```
factory.setReconnAttemptCount(10);  
factory.setReconnAttemptDelay(1000);  
factory.setReconnAttemptTimeout(1000);
```

Configuring Clients for Unshared State Connections



Unshared state failover is an extension of the JMS specification. Because state is not shared among servers, messages can be lost, duplicated, or delivered out-of-order across the failover process.

Unshared state connections are created differently from shared state connections in several important ways:

- For Java applications, a JAR file must be present in the environment CLASSPATH of the client.
- For C applications, a header file must be included and clients must link using the unshared state library.
- The connection must be created using an unshared state connection factory.
- The server URLs must be specified using unshared state syntax.

Include the Unshared State Library

Java Applications Before creating the connection factory, ensure that the CLASSPATH includes the JAR file:

`tibjmsufo.jar`

C Applications Include the `tibemsufo.h` header file.

Create an Unshared State Connection Factory

To create unshared state connections, use the relevant methods:

Java Applications `java com.tibco.tibems.ufo package.`

C Applications `tibemsufo` library and functions.

Connection Recovery

When an unshared state connection fails, the connection's `ExceptionListener` callback is invoked. To recover the connection—repair it so that it is connected to an active server—the client application calls the connection factory's `recoverConnection` method or `tibemsUFOConnectionFactory_RecoverConnection` function. This must be

performed in the `ExceptionListener` callback. The `recover` connection method blocks until the connection (and its related objects, including sessions, producers, and consumers) are fully recreated, or until it has failed in all its attempts to recreate these objects.

As long as the unshared state client has a valid connection, the API behaves the same as the standard EMS client. However, when the unshared state client's connection is broken, the API performs as follows:

1. Methods called inside a `MessageListener` callback immediately return a Java exception `ConnectionFactoryException` or C status of `TIBEMS_SERVER_NOT_CONNECTED`.
2. Methods called elsewhere block until the connection is valid again.

Note that the connection is considered broken from the point where the underlying TCP/SSL connection fails, and until `recoverConnection` or `tibemsConnectionFactory.RecoverConnection` successfully returns.

Specify Server URLs

When a server connection is lost during an unshared state failover, clients attempt to reconnect to the second server. To enable a client to reconnect, you must specify the URLs of both servers when creating a connection.

- **Unshared State** Specify multiple servers as a list of URLs separated by plus (+) signs. For example, to identify the first server as `tcp://server0:7222`, and the second server as `tcp://server1:7344`, you can specify:
`serverUrl=tcp://server0:7222+tcp://server1:7344`
- **Dual State** To combine shared state server pairs with unshared state servers, use commas to separate the servers that share state, and plus (+) signs to separate servers that do not share state. For example, this line specifies server `a1` and `a2` as a fault-tolerant pair that share state, and servers `b1` and `b2` as a second pair with shared state:

```
serverUrl=tcp://a1:8222,tcp://a2:8222+tcp://b1:8222,tcp://b2:8222
```

Note that `a1` and `a2` do not share state with `b1` and `b2`.

The client attempts to connect to each URL in the order listed. If a connection to one URL fails, the client tries the next URL in the list. The client tries the URLs in sequence until all URLs have been tried. If the first failed connection was not the first URL in the list, the attempts wrap to the start of the list (so each URL is tried). If none of the attempts succeed, the connection fails.



Server lookup functions do not permit unshared state syntax. That is, you cannot separate server URLs using the plus (+) symbol during a server lookup.

Chapter 20 **Working With Routes**

This chapter describes routing of messages among TIBCO Enterprise Message Service servers.

Topics

- [Overview of Routing, page 510](#)
- [Route, page 511](#)
- [Zone, page 514](#)
- [Active and Passive Routes, page 517](#)
- [Configuring Routes and Zones, page 518](#)
- [Routed Topic Messages, page 523](#)
- [Routed Queues, page 528](#)
- [Routing and Authorization, page 531](#)

Overview of Routing

TIBCO Enterprise Message Service servers can route messages to other servers.

- Topic messages can travel one hop or multiple hops (from the first server).
- Queue messages can travel only one hop *to* the home queue, and one hop *from* the home queue.

You can define routes using an administrative interface (that is, configuration files, `tibemsadmin`, or administration APIs).

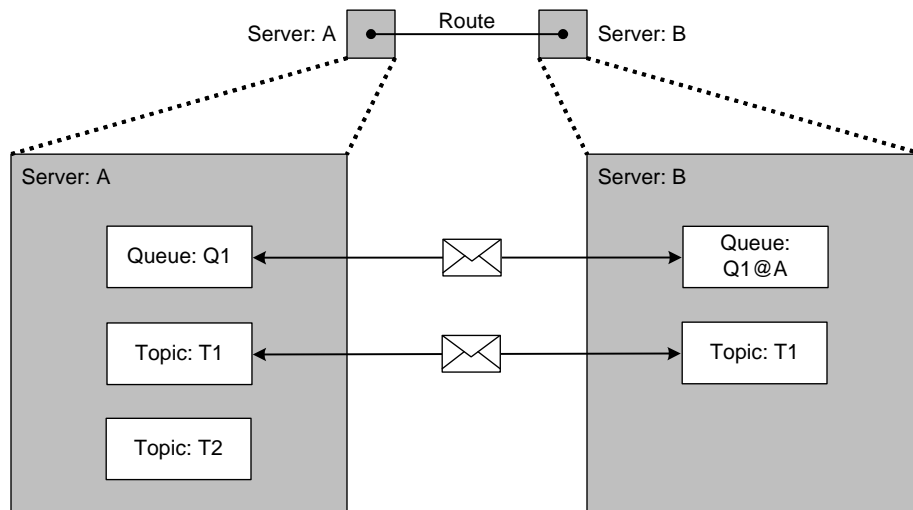
Route

Basic Operation

- Each *route* connects two TIBCO Enterprise Message Service servers.
- Each route forwards messages between corresponding destinations (that is, global topics with the same name, or explicitly routed queues) on its two servers.
- Routes are bidirectional; that is, each server in the pair forwards messages along the route to the other server.

For example, the compact view at the top of [Figure 29](#) denotes a route between two servers, A and B. The exploded view beneath it illustrates the behavior of the route. Each server has a global topic named T1, and a routed queue Q1; these destinations correspond, so the route forwards messages between them. In addition, server A has a global topic T2, which does not correspond to any topic on server B. The route does not forward messages from T2.

Figure 29 Routes: bidirectionality and corresponding destinations

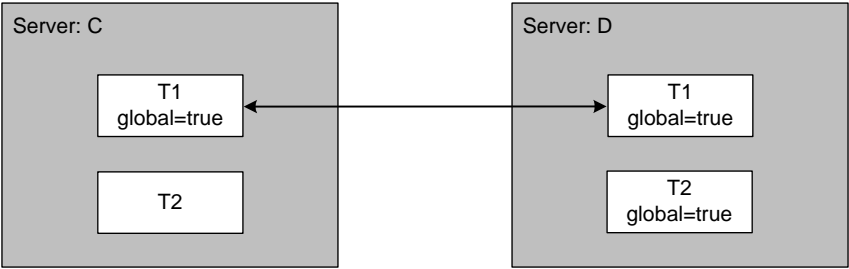


Global Destinations

Routes forward messages only between global destinations—that is, for topics the `global` property must be set on both servers (for queues, see [Routed Queues on page 528](#)). (For more information about destination properties, See [Destination Properties on page 58](#).)

[Figure 30](#) illustrates a route between two servers, C and D, with corresponding destinations T1 and T2. Notice that T1 is global on both C and D, so both servers forward messages across the route to the corresponding destination. However, T2 is not global on C, neither C nor D forward T2 messages to one another.

Figure 30 Routes: global destinations



Unique Routing Path

It is illegal to define a set of routes that permit a message to reach a server by more than one path. TIBCO Enterprise Message Service servers detect illegal duplicate routes and report them as configuration errors.

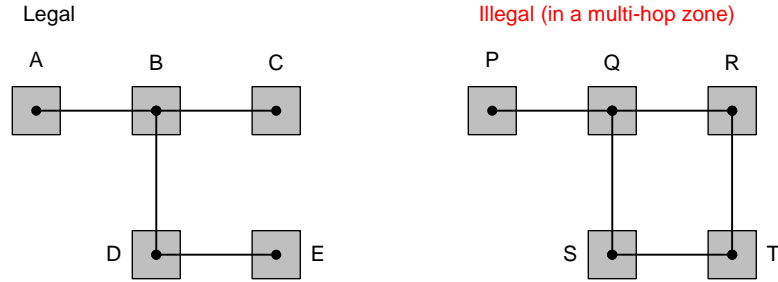
[Figure 31 on page 513](#) depicts two sets of routes. On the left, the routes connecting servers A, B, C, D and E form an acyclic graph, with only one route connecting any pair of servers; this configuration is legal (in any zone).

In contrast, the routing configuration on the right is illegal in a multi-hop zone. The graph contains redundant routing paths between servers Q and S (one direct, and one through R and T).



Note that the configuration on the right is illegal only in a multi-hop zone; it is legal in a one-hop zone. For further information, see [Zone on page 514](#).

Figure 31 Routes: Unique Path



Zone

Zones restrict the behavior of routes, so you can configure complex routing paths. Zones affect topic messages, but not queue messages.

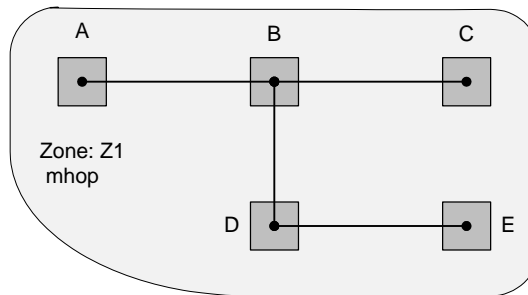
Basic Operation

A *zone* is a named set of routes. Every route belongs to a zone. A zone affects the forwarding behavior of its routes:

- In a multi-hop (`mhop`) zone, topic messages travel along all applicable routes to all servers connected by routes within the zone.
- In a one-hop (`1hop`) zone, topic messages travel only one hop (from the first server).
- Queue messages travel only one hop, even within multi-hop zones.

For example, [Figure 32](#) depicts a set of servers connected by routes within a multi-hop zone, Z1. If a client sends a message to a global topic on server B, the servers forward the message to A, C, D and E (assuming there are subscribers at each of those servers). In contrast, if Z1 were a one-hop zone, B would forward the message to A, C and D—but D would *not* forward it E.

Figure 32 Zones: multi-hop



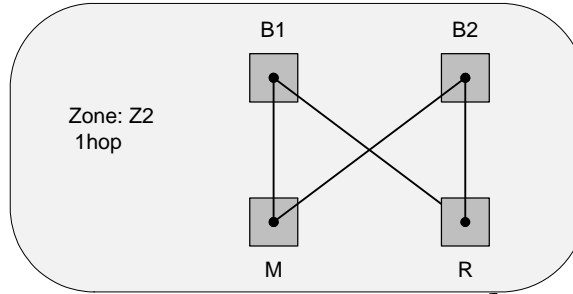
Eliminating Redundant Paths with a One-Hop Zone

[Figure 33](#) illustrates an enterprise with four servers:

- B1 and B2 serve producers at branch offices of an enterprise.
- M serves consumers at the main office, which process the messages from the branches.
- R serves consumers that record messages for archiving, auditing, and backup.

The goal is to forward messages from B1 and B2 to both M and R. The routing graph *seems* to contain a cycle—the path from B1 to M to B2 to R duplicates the route from B1 to R. However, since these routes belong to the one-hop zone Z2, it is impossible for messages to travel the longer path. Instead, this limitation results in the desired result—forwarding from B1 to M and R, and from B2 to M and R.

Figure 33 Zones: one-hop



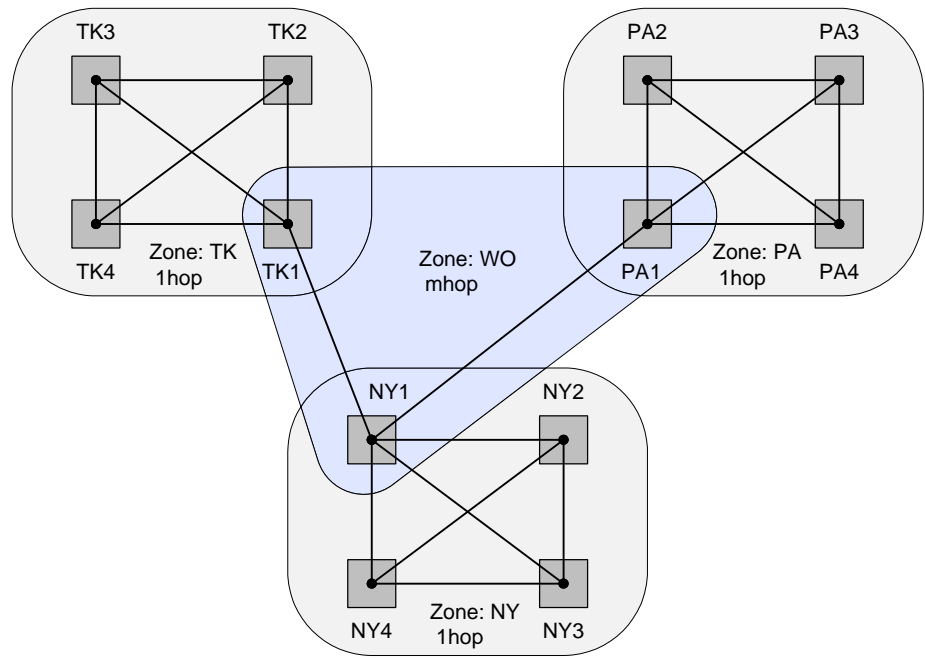
Overlapping Zones

A server can have routes that belong to several zones. When zones overlap at a server, the routing behavior within each zone does not limit routing in other zones. That is, when a forwarded message reaches a server with routes in several zones, the message crosses zone boundaries, and its hop count is reset to zero.

[Figure 34 on page 516](#) illustrates an enterprise with one-hop zones connecting all the servers in each of several cities in a fully-connected graph. Zone TK connects all the servers in Tokyo; zone NY connects all the servers in New York; zone PA connects all the servers in Paris. In addition, the multi-hop zone WO connects one server in each city.

When a client of server TK3 produces a message, it travels one hop to each of the other Tokyo servers. When the message reaches TK1, it crosses into zone WO. TK1 forwards the message to NY1, which in turn forwards it to PA1. When the message reaches NY1, it crosses into zone NY (with hop count reset to zero); NY1 forwards it one hop to each of the other New York servers. Similarly, when the message reaches PA1, it crosses into zone PA (with hop count reset to zero); PA1 forwards it one hop to each of the other Paris servers.

Figure 34 Zones: overlap



Active and Passive Routes

A route connects two servers. You may configure a route at either or both of the servers.

Active-Passive Routes

When you configure a route at only one server, this asymmetry results in two perspectives on the route:

- A route is *active* from the perspective of the server where it is configured. This server actively initiates the connection to the other server, so we refer to it as the *active server*, or *initiating server*.
- A route is *passive* from the perspective of the other server. This server passively accepts connection requests from the active server, so we refer to it as the *passive server*.

A server can have both active and passive routes. That is, you can configure server S to initiate routes, and also configure other servers to initiate routes to S.

You can specify and modify the properties of an active route, but not those of a passive route. That is, properties of routes are associated with the server where the route is configured, and which initiates the connection.



Note that defining a route specifies a zone as well (either implicitly or explicitly). The first route in the zone defines the type of the route; subsequent routes in the same zone must have the same zone type (otherwise, the server reports an error).

Active-Active Routes

Two servers can both configure an active route one to the other. This arrangement is called an *active-active configuration*. For example, server A specifies a route to server B, and B specifies a route to A. Either server can attempt to initiate the connection. This configuration results in only one connection; it does *not* result in redundant routes.

You can promote an *active-passive* route to an *active-active* route. To promote a route, use this command on the passive server:

```
create route name url=url
```

The *url* argument is required, so that the server (where the route is being promoted) can connect to the other server if the route becomes disconnected. See also [create route on page 132](#).

The promoted route behaves as a statically configured route—that is, it persists messages for durable subscribers, and stores its configuration in [routes.conf](#), and administrators can modify its properties.

Configuring Routes and Zones

You can create routes using the administration tool (see [Chapter 6 on page 123](#)), or the administration APIs (see `com.tibco.tibjms.admin.RouteInfo` in the online documentation).

Syntax To create a route using the administration tool, first connect to one of the servers, then use the `create route` command with the following syntax:

```
create route name url=URL zone_name=zone_name zone_type=1hop|1mhop properties
```

- *name* is the name of the server at the other end of the route; it also becomes the name of the route.
- *URL* specifies the other server by its URL—including protocol and port.

If your environment is configured for fault tolerance, the URL can be a comma-separated list of URLs denoting fault-tolerant servers. For more information about fault tolerance, see [Chapter 19, Fault Tolerance, on page 485](#).

- *zone_name* specifies that the route belongs to the routing zone with this name. When absent, the default value is `default_mhop_zone` (this default yields backward compatibility with configurations from releases earlier than 4.0).
- The zone type is either `1hop` or `mhop`. When omitted, the default value is `mhop`.
- *properties* is a space-separated list of properties for the route. Each property has the syntax:

prop_name=value

For gating properties that control the flow of topics along the route, see [Selectors for Routing Topic Messages on page 525](#).

For properties that configure the Secure Sockets Layer (SSL) protocol for the route, see [Routing and SSL on page 519](#).

Example For example, these commands on server A would create routes to servers B and C. The route to B belongs to the one-hop zone Z1. The route to C belongs to the multi-hop zone ZM.

```
create route B url=tcp://B:7454 zone_name=Z1 zone_type=1hop
create route C url=tcp://C:7455 zone_name=ZM zone_type=mhop
```


Show Routes You can display these routes using the `show routes` command in the administration tool:

```
show routes
Route  T    ConnID  URL                Zone    T
B      A      3    tcp://B:7454       Z1      1
C      A      -    tcp://C:7455       ZM      m
```

- The `Route` column lists the name of the passive server.
- The `T` column indicates whether the route is active (A) or passive (P), from the perspective of server A.
- The `ConnID` column contains either an integer connection ID (if the route is currently connected, or a dash (-) if the route is not connected.

Routes to Fault-Tolerant Servers

You can configure servers for fault tolerance. Client applications can specify the primary and backup servers; if the client's connection to the primary server fails, the client can connect to the backup server and resume operation. Similarly, a route specification can specify primary and secondary passive servers, so that if the route to the primary server fails, the active server can connect to the backup server and resume routing.

Failover behavior for route connections is similar to that for client connections; see [Configuring Clients for Fault-Tolerant Connections on page 504](#).

Example

```
create route B url=tcp://B:7454,tcp://BBackup:7454 zone_name=Z1 zone_type=1hop
```

Routing and SSL

When configuring a route, you can specify SSL parameters for the connection. Although both participants in an SSL connection must specify a similar set of parameters, each server specifies this information in a different place:

- The passive server must specify SSL parameters in its main configuration file, `tibemsd.conf`.
- When a server initiates an SSL connection, it sends the route's SSL parameters to identify and authenticate itself to the passive server. You can specify these parameters when creating the route, or you can specify them in the route configuration file, `routes.conf`.

Table 80 lists the parameters that you can specify in the `routes.conf` configuration file, or on the command line when creating a route. The parameters for configuring SSL between routed servers are similar to the parameters used to configure SSL between server and clients; see [Chapter 18, Using the SSL Protocol, on page 465](#).

Table 80 SSL Parameters for Routes (Sheet 1 of 3)

Parameter	Description
<code>ssl_identity</code>	<p>The server’s digital certificate in PEM, DER, or PKCS#12 format. You can copy the digital certificate into the specification for this parameter, or you can specify the path to a file that contains the certificate in one of the supported formats.</p> <p>For more information, see File Names for Certificates and Keys on page 469.</p>
<code>ssl_issuer</code>	<p>Certificate chain member for the server. Supply the entire chain, including the CA root certificate. The server reads the certificates in the chain in the order they are presented in this parameter.</p> <p>The certificates must be in PEM, DER, PKCS#7 or PKCS#12 format.</p> <p>Example</p> <pre>ssl_issuer = certs\CA_root.pem ssl_issuer = certs\CA_child1.pem ssl_issuer = certs\CA_child2.pem</pre> <p>For more information, see File Names for Certificates and Keys on page 469.</p>
<code>ssl_private_key</code>	<p>The local server’s private key. If the digital certificate in <code>ssl_identity</code> already includes this information, then you may omit this parameter.</p> <p>This parameter accepts private keys in PEM, DER and PKCS#12 formats.</p> <p>You can specify the actual key in this parameter, or you can specify a path to a file that contains the key.</p> <p>For more information, see File Names for Certificates and Keys on page 469.</p>

Table 80 SSL Parameters for Routes (Sheet 2 of 3)

Parameter	Description
ssl_password	<p>Private key or password for private keys.</p> <p>You can set passwords using the <code>tibemsadmin</code> tool. When passwords are set with this tool, the password is obfuscated in the configuration file. For more information, see Chapter 6, Using the EMS Administration Tool, on page 123.</p>
ssl_trusted	<p>List of certificates that identify trusted certificate authorities.</p> <p>The certificates must be in PEM, DER or PKCS#7 format. You can either provide the actual certificates, or you can specify a path to a file containing the certificate chain.</p> <p>For more information, see File Names for Certificates and Keys on page 469.</p>
ssl_verify_host	<p>Specifies whether the server must verify the other server's certificate. The values for this parameter are <code>enabled</code> and <code>disabled</code>.</p> <p>When omitted, the default is <code>enabled</code>, signifying the server must verify the other server's certificate.</p> <p>When this parameter is <code>disabled</code>, the server establishes secure communication with the other server, but does not verify the server's identity.</p>

Table 80 SSL Parameters for Routes (Sheet 3 of 3)

Parameter	Description
ssl_verify_hostname	<p>Specifies whether the server must verify the name in the CN field of the other server’s certificate. The values for this parameter are enabled and disabled.</p> <p>When omitted, the default is enabled, signifying the server must verify the name of the connected host or the name specified in the ssl_expected_hostname parameter against the value in the server’s certificate. If the names do not match, the connection is rejected.</p> <p>When this parameter is disabled, the server establishes secure communication with the other server, but does not verify the server’s name.</p>
ssl_expected_hostname	<p>Specifies the name expected in the CN field of the other server’s certificate. If this parameter is not set, the default is the hostname of the other server.</p> <p>This parameter is relevant only when the ssl_verify_hostname parameter is enabled.</p>
ssl_ciphers	<p>Specifies a list of cipher suites, separated by colons (:).</p> <p>This parameter accepts both the OpenSSL name for cipher suites, or the longer descriptive names.</p> <p>For information about available cipher suites and their names, see Specifying Cipher Suites on page 476.</p>
ssl_rand_egd	<p>The path for the installed entropy gathering daemon (EGD), if one is installed. This daemon generates random numbers.</p>

Routed Topic Messages

A server forwards topic messages along routes only when the `global` property is defined for the topic; see [addprop topic on page 129](#) and [create topic on page 133](#).

Topic messages can traverse multiple hops.

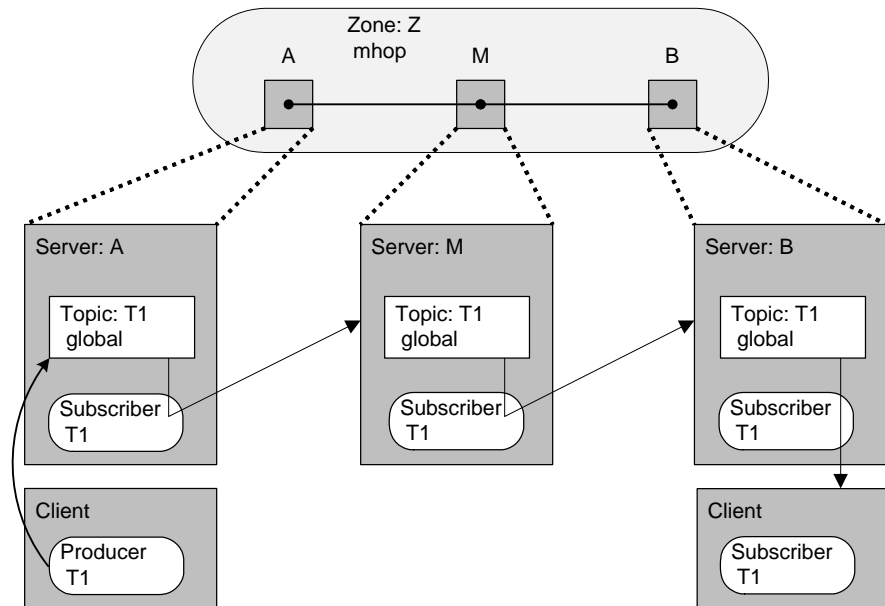
When a route becomes disconnected (for example, because of network problems), the forwarding server stores topic messages. When the route reconnects, the server forwards the stored messages.

Servers connected by routes do exchange messages sent to temporary topics.

Propagating Registered Interest

To ensure forwarding of messages along routes, servers propagate their topic subscriptions to other servers. For example, the top of [Figure 35](#) depicts an enterprise with three servers—A, M and B—connected by routes in a multi-hop zone. The bottom of [Figure 35](#) illustrates the mechanism at work within the servers to route messages from a producer client of server A, through server M, to server B and its subscriber client. Consider this sequence of events.

Figure 35 Routing: Propagating Subscribers



1. All three servers configure a global topic T1.
2. At bottom right of [Figure 35](#), a client of server B creates a subscriber to T1.
3. Server B, registers interest in T1 on behalf of the client by creating an internal subscriber object.
4. Because a route connects servers M and B, server B propagates its interest in T1 to server M. In response, M creates an internal subscriber to T1 on behalf of server B. This subscriber ensures that M forwards (that is, delivers) messages from topic T1 to B. Server B behaves as a client of server M.
5. Similarly, because a route connects servers A and M, server M propagates its interest in T1 to server A. In response, A creates an internal subscriber to T1 on behalf of server M. This subscriber ensures that A forwards messages from topic T1 to M. Server M behaves as a client of server A.
6. When a producer client of server A sends a message to topic T1, A forwards it to M. M accepts the message on its topic T1, and forwards it to B. B accepts the message on its topic T1, and passes it to the client.

Subscriber Client Exit	<p>If the client of server B creates a <i>non-durable</i> subscriber to T1, then if the client process exits, the servers delete the entire sequence of internal subscribers. When the client restarts, it generates a new sequence of subscribers; meanwhile, the client might have missed messages.</p>
	<p>If the client of server B creates a <i>durable</i> subscriber to T1, then if the client process exits, the entire sequence of internal subscribers remains intact; messages continue to flow through the servers in store-and-forward fashion. When the client restarts, it can consume all the messages that B has stored in the interim.</p>
Server Failure	<p>In an active-active route between servers B and M, if B fails, then M retains its internal subscriber and continues to store messages for clients of B. When B reconnects, M forwards the stored messages.</p>
	<p>In an active-passive route configured on B, if B fails, then M removes its internal subscriber and does not store messages for clients of B—potentially resulting in a gap in the message stream. When B reconnects, M creates a new internal subscriber and resumes forwarding messages.</p>
	<p>In an active-passive route configured on A, if either server fails, then M retains its internal subscriber in the same way as an active-active route. However, B does not retain its internal state which it uses to suppress duplicate messages from A and can deliver messages to its consumers after they have consumed them. Therefore, if it is desirable to not lose messages and to not have duplicate messages, the route should be active-active.</p>

Network Failure	If an active-passive connection between B and M is disrupted, M displays the same behavior as during a server failure.
maxbytes	Combining durable subscribers with routes creates a potential demand for storage—especially in failure situations. For example, if server B fails, then server M stores messages until B resumes. We recommend that you set the <code>maxbytes</code> or <code>maxmsgs</code> property of the topic (T1) on each server, to prevent unlimited storage growth (which could further disrupt operation).

Selectors for Routing Topic Messages

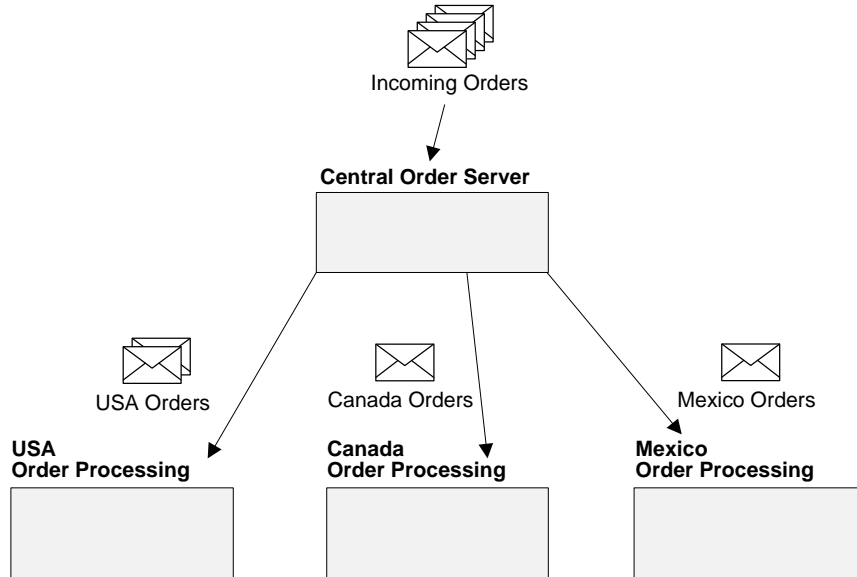
Motivation	A server forwards a global topic message along routes to all servers with subscribers for that topic. When each of those other servers requires only a small subset of the messages, this policy could potentially result in a high volume of unwanted network traffic. You can specify <i>message selectors</i> on routes to narrow the subset of topic messages that traverse each route.
------------	---



Message selectors on routes are different from message selectors on individual subscribers, which narrow the subset of messages that the server delivers to the subscriber client.

Example	Figure 36 on page 525 illustrates an enterprise with a central server for processing customer orders, and separate regional servers for billing those orders. For optimal use of network capacity, we configure topic selectors so that each regional server gets only those orders related to customers within its region.
---------	---

Figure 36 Routing: Topic Selectors, example

**Specifying Selectors**

Specify message selectors for global topics as properties of routes. You can define these properties in two ways:

- Define selectors when creating the route (either in `routes.conf`, or with the administrator command `create route`).
- Manipulate selectors on an existing route (using the `addprop`, `setprop`, or `removeprop` administrator commands).



If you change the message selectors on a route, only incoming messages are evaluated against the new selectors. Messages pending in the server are re-evaluated only if the server is restarted.

Syntax

The message selector properties have the same syntax whether they appear in a command or in a configuration file:

```
incoming_topic=topicName selector="msg-selector"
outgoing_topic=topicName selector="msg-selector"
```



The terms *incoming* and *outgoing* refer to the perspective of the active server—where the route is defined.

topicName is the name of a global topic.

msg-selector is a message selector string. For detailed information about message selector syntax, see the documentation for class `Message` in *TIBCO Enterprise Message Service Java API Reference*.

Example Syntax In the example of [Figure 36](#), an administrator might configure these routes on the central order server:

```
setprop route Canada outgoing_topic="orders" selector="country='Canada'"
setprop route Mexico outgoing_topic="orders" selector "country='Mexico'"
setprop route USA outgoing_topic="orders" selector="country='USA'"
```

Those commands would create these entries in `routes.conf`:

```
[Canada]
url=ssl://canada:7222
outgoing_topic=orders selector="country='Canada'"
...
[Mexico]
url=ssl://mexico:7222
outgoing_topic=orders selector="country='Mexico'"
...
[USA]
url=ssl://usa:7222
outgoing_topic=orders selector="country='USA'"
...
```

Symmetry `outgoing_topic` and `incoming_topic` are symmetric. Whether A specifies a route to B with `incoming_topic` selectors, or B specifies a route to A with `outgoing_topic` selectors, the effect is the same. That is, B sends only those messages that match the selector over the route.

Active-Active Configuration In an active-active configuration, you may specify selectors on either or both servers. If you specify `outgoing_topic` selector S1 for topic T on server A, and `incoming_topic` selector S2 for T on server B, then the effective selector for T on the route from A to B is (S1 AND S2).

See also [Active and Passive Routes on page 517](#).

Wildcards You can specify wildcards in topic names. For each actual topic, the server uses logical AND to combine all the selectors that match the topic.



However, routing of topic messages is only reliably supported when the subscriber's topic is a subset (or equal) of the configured global topic. Similarly, intersections are not supported. For example, if `topics.conf` contains `foo.*` and `foo.a*`, the following subscriptions are correct:

```
foo.*
foo.1
bar.a.b
```

The following subscriptions are *not* correct:

```
foo.>
bar.*.b
```

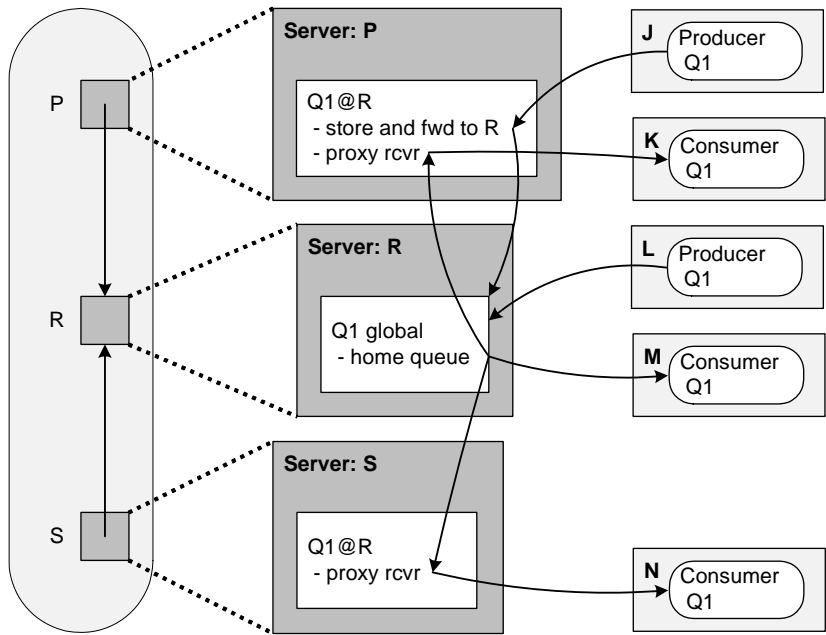
Routed Queues

With respect to routing, queues differ from topics in several respects:

- Servers route queue messages between the queue owner and adjacent servers.
- The concept of zones and hops does not apply to queue messages (only to topic messages).

The left side of [Figure 37](#) depicts an enterprise with three servers—P, R and S—connected by routes. The remainder of [Figure 37](#) illustrates the mechanisms that routes queue messages among servers (center) and their clients (right side).

Figure 37 Routing: Queues



Owner & Home Server R defines a global queue named Q1. R is the *owner* of Q1.

Servers P and S define *routed queues* Q1@R. This designation indicates that these queues depend upon and reflect their *home queue* (that is, Q1 on server R). Notice that the designation Q1@R is only for the purpose of configuration; clients of P refer to the routed queue as Q1.

- Example** When J sends a message to Q1, server P forwards the message to the home queue—Q1 on server R.
- Now the message is available to receivers on all three servers, P, R and S—although only one client can consume the message. Either Q1 on P receives it on behalf of K; or Q1 on S receives it on behalf of N; or M receives it directly from the home queue.
- Producers** From the perspective of producer clients, a routed queue stores messages and forwards them to the home queue. For example, when J sends a message to Q1 on server P, P forwards it to the queue owner, R, which delivers it to Q1 (the home queue).
- The message is not available for consumers until it reaches the home queue. That is, client K cannot consume the message directly from server P.
- If server R fails, or the route connection from P to R fails, P continues to store messages from K in its queue. When P and R resume communication, P delivers the stored messages to Q1 on R.
- Similarly, routed queues do not generate an exception when the `maxbytes` and `maxmsgs` limits are exceeded in the routed server. Clients can continue to send messages to the queue after the limit is reached, and the messages will be stored in the routed server until the error condition is cleared.
- Consumers** From the perspective of consumer clients, a routed queue acts as a proxy receiver. For example, when L sends a message to Q1 on server R, Q1 on P can receive it from R on behalf of K, and immediately gives it to K.
- If server P fails, or the route connection from P to R fails, K cannot receive messages from Q1 until the servers resume communication. Meanwhile, M and N continue to receive messages from Q1. When P and R resume communication, K can again receive messages through Q1 on P.



Receiving messages from a routed queue using either a small timeout (less than one second) or no wait can cause unexpected behavior. A small timeout value increases the chances that protocol messages may not be processed correctly between the routed servers. For example, queue receivers may not be correctly destroyed.

- Configuration** You must explicitly configure each routed queue in `queues.conf`—clients cannot create routed queues dynamically.



Dynamic routed queues are not supported. In a future release, the server will consider a routed queue with a wildcard as a misconfiguration and will fail to start when `startup_abort_list` includes `CONFIG_ERRORS`.

You may use the administration tool or API to configure routed queues; see [addprop queue on page 128](#) and [create queue on page 132](#).

To configure a routed queue, specify the queue name and the server name of the queue owner; for example, on server P, configure:

Q1@R



It is legal to use this notation even for the home queue. The queue owner recognizes its own name, and ignores the location designation (@R).

It is illegal to configure a routed queue as `exclusive`.

Browsing Queue browsers cannot examine routed queues. That is, you can create a browser only on the server that owns the home queue.

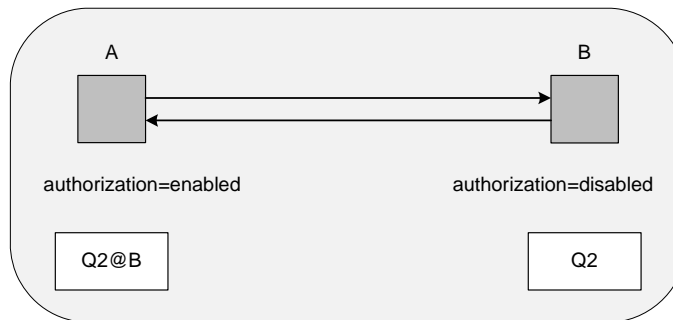
Transactions TIBCO Enterprise Message Service does not support transactional consumers on routed queues (through the use of XA or local transacted sessions).

Routing and Authorization

User & Password

When a server's `authorization` parameter is enabled, other servers that actively connect to it must authenticate themselves by name and password, or by X.509 certificate.

Figure 38 Routing: Authorization



In [Figure 38](#), servers A and B both configure active routes to one another.

- Because A enables authorization, A must configure a user named B.
- However, because B disables authorization, A need not identify itself to B, and B need not configure a user named A.

ACL

When routing a secure topic or queue, servers consult the ACL specification before forwarding each message. The servers must grant one another appropriate permissions to send, receive, publish or subscribe.

For example, in [Figure 38](#), you don't need an ACL for messages to flow from A (where a producer is sending to) to B (where a consumer is consuming from) because B has authorization turned off and messages are being sent to and consumed from queues. However, if messages were to flow from B to A (producer connects to B and consumer connects to A), then server A's ACL should grant user B send permission on the queue Q2.

If we were to use topics in this example, then for messages to flow from A to B, you would need A to grant B the `subscribe` and `durable` permission on the topic (`global` on both servers). And for messages to flow from B to A, you would have to grant topic B `publish` permission on the topic.

See Also [Chapter 8, Authentication and Permissions, on page 265](#)

Appendix A **Monitor Messages**

This appendix lists all topics on which the server publishes messages for system events. The message properties for messages published on each topic are also described. See [Monitoring Server Events on page 454](#) for more information about monitor topics and messages.

Topics

- [Description of Monitor Topics, page 534](#)
- [Description of Topic Message Properties, page 537](#)

Description of Monitor Topics

Table 81 describes each monitor topic.

Table 81 Monitor topics

Topic	Message Is Published When...
<code>\$sys.monitor.admin.change</code>	The administrator has made a change to the configuration.
<code>\$sys.monitor.connection.connect</code>	A user attempts to connect to the server.
<code>\$sys.monitor.connection.disconnect</code>	A user connection is disconnected.
<code>\$sys.monitor.connection.error</code>	An error occurs on a user connection.
<code>\$sys.monitor.consumer.create</code>	A consumer is created.
<code>\$sys.monitor.consumer.destroy</code>	A consumer is destroyed.
<code>\$sys.monitor.flow.engaged</code>	Stored messages rise above a destination's limit, engaging the flow control feature.
<code>\$sys.monitor.flow.disengaged</code>	Stored messages fall below a destination's limit, disengaging the flow control feature.
<code>\$sys.monitor.limits.connection</code>	Maximum number of hosts or connections is reached.
<code>\$sys.monitor.limits.queue</code>	Maximum bytes for queue storage is reached.
<code>\$sys.monitor.limits.server</code>	Server memory limit is reached.
<code>\$sys.monitor.limits.topic</code>	Maximum bytes for durable subscriptions is reached.
<code>\$sys.monitor.multicast.stats</code>	The message published contains low-level PGM statistics from the server and multicast daemons.
<code>\$sys.monitor.multicast.status</code>	A message consumer subscribes or attempts to subscribe to a multicast-enabled topic.
<code>\$sys.monitor.producer.create</code>	A producer is created.
<code>\$sys.monitor.producer.destroy</code>	A producer is destroyed.
<code>\$sys.monitor.queue.create</code>	A dynamic queue is created.

Table 81 *Monitor topics*

Topic	Message Is Published When...
<code>\$sys.monitor.route.connect</code>	A route connection is attempted.
<code>\$sys.monitor.route.disconnect</code>	A route connection is disconnected.
<code>\$sys.monitor.route.error</code>	An error occurs on a route connection.
<code>\$sys.monitor.route.interest</code>	A change in registered interest occurs on the route.
<code>\$sys.monitor.server.info</code>	The server sends information about an event; for example, a log file is rotated.
<code>\$sys.monitor.server.warning</code>	The primary server detects a disconnection from the backup server.
<code>\$sys.monitor.topic.create</code>	A dynamic topic is created.
<code>\$sys.monitor.tx.action</code>	A local transaction commits or rolls back.
<code>\$sys.monitor.xa.action</code>	An XA transaction commits or rolls back.

Table 81 Monitor topics

Topic	Message Is Published When...
<code>\$sys.monitor.D.E.destination</code>	<p>A message is handled by a destination. The name of this monitor topic includes two qualifiers (<i>D</i> and <i>E</i>) and the name of the destination you wish to monitor.</p> <p><i>D</i> signifies the type of destination and whether to include the entire message:</p> <ul style="list-style-type: none">• <i>T</i> — topic, include full message (as a byte array) into each event• <i>t</i> — topic, do not include full message into each event• <i>Q</i> — queue, include full message (as a byte array) into each event• <i>q</i> — queue, do not include full message into each event <p><i>E</i> signifies the type of event:</p> <ul style="list-style-type: none">• <i>r</i> for receive• <i>s</i> for send• <i>a</i> for acknowledge• <i>p</i> for premature exit of message• <i>*</i> for all event types <p>For example, <code>\$sys.monitor.T.r.corp.News</code> is the topic for monitoring any received messages to the topic named <code>corp.News</code>. The message body of any received messages is included in monitor messages on this topic. The topic <code>\$sys.monitor.q.*.corp.*</code> monitors all message events (send, receive, acknowledge) for all queues matching the name <code>corp.*</code>. The message body is not included in this topic's messages.</p> <p>The messages sent to this type of monitor topic include a description of the event, information about where the message came from (a producer, route, external system, and so on), and optionally the message body, depending upon the value of <i>D</i>.</p> <p>See Monitoring Messages on page 454 for more information about message monitoring.</p>

Description of Topic Message Properties

[Table 82](#) describes the properties that monitor topic messages can contain. Each monitor message can have a different set of these properties.

Table 82 Message properties

Property	Contents
conn_connid	Connection ID of the connection that generated the event.
conn_ft	Whether the client connection is a connection to a fault-tolerant server.
conn_hostname	Hostname of the connection that generated the event.
conn_ssl	Whether the connection uses the SSL protocol.
conn_type	Type of connection that generated the event. This property can have the following values: <ul style="list-style-type: none"> • Admin • Topic • Queue • Generic • Route • FT (connection to fault-tolerant server) • Unknown
conn_username	User name of the connection that generated the event.
conn_xa	Whether the client connection is an XA connection.
event_action	The action that caused the event. This property can have the values listed in Table 83 on page 542 .
event_class	The type of monitoring event (that is, the last part of the topic name without the <code>\$sys.monitor</code>). For message monitoring, the value of this property is always set to <code>message</code> .
event_description	A text description of the event that has occurred.

Table 82 Message properties

Property	Contents
event_reason	The reason the event occurred (usually an error). The values this property can have are described in Table 84 on page 544 .
event_route	For routing, the route that the event occurred on.
message_bytes	When the full message is to be included for message monitoring, this field contains the message as a byte array. You can use the <code>createFromBytes</code> method (in the various client APIs) to recover the message.
mode	Message delivery mode. This values of this property can be the following: <ul style="list-style-type: none">• <code>persistent</code>• <code>non_persistent</code>• <code>reliable</code>
msg_id	Message ID.
msg_seq	Message sequence number.
msg_size	Message size, in bytes.
msg_timestamp	Message timestamp.
msg_expiration	Message expiration.
replyTo	Message JMSReplyTo.
rv_reply	Message RV reply subject.
source_id	ID of the source object.

Table 82 Message properties

Property	Contents
source_name	<p>Name of the source object involved with the event. This property can have the following values:</p> <ul style="list-style-type: none"> • <code>XID</code> (global transaction ID) • <code>message_id</code> • <code>connections</code> (number of connections) • <code>unknown</code> (unknown name) • Any server property name • the name of the user, or <code>anonymous</code>
source_object	<p>Source object that was involved with the event. This property can have the following values:</p> <ul style="list-style-type: none"> • <code>producer</code> • <code>consumer</code> • <code>topic</code> • <code>queue</code> • <code>permissions</code> • <code>durable</code> • <code>server</code> • <code>transaction</code> • <code>user</code> • <code>group</code> • <code>connection</code> • <code>message</code> • <code>jndiname</code> • <code>factory</code> • <code>file</code> • <code>limits</code> (a limit, such as a memory limit) • <code>route</code> • <code>transport</code>

Table 82 Message properties

Property	Contents
source_value	Value of source object.
stat_msgs	Message count statistic for producer or consumer.
stat_size	Message size statistic for producer or consumer.
target_admin	Whether the target object is the admin connection.
target_created	Time that the consumer was created (in milliseconds since the epoch).
target_dest_name	Name of the target destination
target_dest_type	Type of the target destination.
target_durable	Name of durable subscriber when target is durable subscriber.
target_group	Group name that was target of the event
target_hostname	Hostname of the target object.
target_id	ID of the target object.
target_channel	Name of the multicast channel.
target_name	<p>Name of the object that was the target of the event. This property can have the following values:</p> <ul style="list-style-type: none">• <code>XID</code> (global transaction ID)• <code>message_id</code>• <code>connections</code> (number of connections)• <code>unknown</code> (unknown name)• Any server property name• the name of the user, or anonymous• <code>channel</code> (multicast channel)
target_nolocal	NoLocal flag when target is durable subscriber.

Table 82 Message properties

Property	Contents
target_object	<p>The general object that was the target of the event. This property can have the following values:</p> <ul style="list-style-type: none">• producer• consumer• topic• queue• durable• server• transaction• user• group• connection• message• jndiname• factory• file• limits (a limit, such as a memory limit)• route• transport
target_selector	Selector when the target is a consumer.
target_subscription	Subscription of the target object when target is durable subscriber.
target_url	URL of the target object.
target_username	Username of the target object.
target_value	Value of the object that was the target of the event, such as the name of a topic, queue, and so on.

Table 83 Event Action Property Values

Event Action Value	Description
accept	connection accepted
acknowledge	message is acknowledged
add	user added to a group
admin_commit	administrator manually committed an XA transaction
admin_rollback	administrator manually rolled back an XA transaction
commit	transaction committed
connect	connection attempted
create	something created
delete	something deleted
disconnect	connection disconnected
flow_engaged	stored messages rise above a destination's limit, engaging the flow control feature
flow_disengaged	stored messages fall below a destination's limit, disengaging the flow control feature
interest	registered interest for a route
modify	something changed
grant	permission granted
premature_exit	message prematurely exited
purge	topic, queue, or durable subscriber purged
receive	message posted into destination
remove	user removed from a group
resume	administrator resumed a route

Table 83 Event Action Property Values

Event Action Value	Description
revoke	permission revoked
rollback	transaction rolled back
rotate_log	log file rotated
send	message sent by server to another party
subscribe	subscription request
suspend	administrator suspended a route
txcommit	administrator manually committed a local transaction
txrollback	administrator manually rolled back a local transaction
xaccommit	an application committed an XA transaction (2-phase)
xaccommit_1phase	an application committed an XA transaction (1-phase)
xastart	an application started a new XA transaction
xastart_join	an application has joined (that is, added) a resource to an existing transaction
xastart_resume	an application resumed a suspended XA transaction
xaend_fail	an application ended an XA transaction, indicating failure
xaend_success	an application ended an XA transaction, indicating success
xaend_suspend	an application suspended an XA transaction
xaprepare	an application prepared an XA transaction
xarecover	an application called recover (to get a list of XA transactions)

Table 83 Event Action Property Values

Event Action Value	Description
xarollback	an application rolled back an XA transaction

Table 84 Event Reason Property Values

Event Reason Value	Description
backup_connected	The fault-tolerant backup server has connected.
backup_disconnected	The fault-tolerant backup server has disconnected.
bridge	Message posted to destination as result of bridging.
closed	Connection was closed.
consumer	For message monitoring, this value signifies a message was sent or acknowledged by a consumer. For all other cases, this value signifies a dynamic topic or queue created for a consumer.
cycle	Cyclic route created.
disabled	Feature not enabled.
discarded	The oldest message on the destination has been discarded to make room for a new message. This occurs when <code>overflowPolicy=discardOld</code> is set on the destination and either the <code>maxmsgs</code> and/or <code>maxbytes</code> limit set for the destination has been exceeded.
duplicate	Duplicate, such as route, global queue or topic.
error	Connection disconnected due to error.
exceeded	Limit exceeded.
expired	Message has been expired by the server.
export	Message exported to a transport.

Table 84 Event Reason Property Values

Event Reason Value	Description
import	Message imported from a transport.
invalid_name	Name not valid, such as route name.
invalid_password	Invalid password provided.
maxredelivery_exceeded	Message has exceeded the maxRedelivery count for the queue.
new_connection	A new connection was established to the server.
not_authorized	Not authorized to perform action.
not_connected	Could not establish connection.
not_found	Something was expected, but not found.
producer	For message monitoring, this value signifies a message was posted by a producer. For all other cases, this value signifies a dynamic topic or queue created for a producer.
reconnect_active	Connection active.
reconnected_connection	The connection to the server has been reestablished.
reconnect_unknown	Connection unknown.
rotatelog	Log file rotated.
route	For message monitoring, this value signifies a message was sent or received from a route. For all other cases, this value signifies a dynamic topic or queue created for a route.
shutdown	Server was shut down.
standby	Server in standby mode.
subscribed	Successful subscription request.
terminated	Connection was terminated.

Appendix B **Error and Status Messages**

This appendix lists all possible error messages that the server can output, alphabetized by category.

Key to this Appendix

Category	The category indicates the general class of error. This appendix is alphabetized by category.
Description	The description explains the error category in more detail.
Resolution	The resolution indicates possible recovery actions that administrators should consider.
Errors	These strings represent all instances of the error, as they appear in EMS server code. Some categories have many error instances; others have only one. These strings can include formatting characters.

Error and Status Messages

Category	A durable consumer was found in the store file for a route that does not exist
Description	On server startup a durable consumer was found in the store file for a route that is not listed in the routes.conf file. This happens if the routes.conf file is manually edited.
Resolution	Make routing changes via administration tools.
Errors	Discarding durable '%s' for route '%s' because the route does not exist.

Category	Admin command failed
Description	An admin tool or program using the admin API attempted an operation that failed for the given reason.
Resolution	The admin tool or admin API provides the failure reason. The user of the tool or API should examine the error and correct the syntax, parameter or configuration that is causing the failure.
Errors	%s: create %s failed: conflicting zone: existing consumer has a different zone %s: create %s failed: detected duplicate durable subscription [%s] for topic [%s]. %s: create %s failed: illegal to use wildcard %s [%s]. %s: create %s failed: invalid %s [%s]. %s: create %s failed: invalid session id=%d. %s: create %s failed: invalid syntax of %s [%s]. %s: create %s failed: invalid temporary %s [%s]. %s: create %s failed: not allowed to create dynamic %s [%s]. Invalid consumer in recover one msg request. Invalid sequence number in recover one msg request.

Category	Authentication error
Description	The EMS server failed to authenticate the user or password.

Resolution	Ensure the user is defined to EMS by one of the methods allowed by the user_auth parameter in the main configuration file. The user is either specified by the application or in the SSL certificate. If the user is defined, reset the password and try again.
Errors	<p>Unable to initialize connection, SSL username error.</p> <p>LDAP authentication failed for user '%s', status = %d</p> <p>LDAP authentication failed for user '%s', no password provided</p>
<hr/>	
Category	Backup server '%s' disconnected
Description	Lost connection with the backup fault-tolerant server.
Resolution	Determine if the backup server is running. If it is running, check for a network partition.
Errors	Backup server '%s' disconnected.
<hr/>	
Category	Bad or missing value for command line parameter
Description	An invalid value was supplied for a command line parameter.
Resolution	Change the value of the named parameter to an acceptable value; for information about tibemsd command line parameters, see EMS documentation.
Errors	<p>'%s' requires an integer argument.</p> <p>'%s' requires a positive integer argument.</p> <p>'%s' requires a string argument.</p>
<hr/>	
Category	Banners and debug traces
Description	Banner and debug traces
Resolution	Not applicable
Errors	<p>%s: %s has been changed</p> <p>%s: %s has been changed to %s</p> <p>%s: %s has been changed to %d</p>

%s: %s has been changed to % PRINTF_LLFMT d

Invalid session for route configuration.

Invalid routed queue information message.

Expired % PRINTF_LLFMT d message%s.

Discarded % PRINTF_LLFMT d message%s.

[%s@%s]: rejected connect from route: invalid password

%s: purged durable '%s'

%s: %s %s '%s' permissions on %s '%s': %s

%s: create %s failed: durable creation access denied for %s [%s].

Async Recs: max=%d avg=%.2f min=%d

Process Id: %d

Server activating on failure of '%s'.

ldap_search_ext_s(%x, %s, %s, %s)

Flow Stall Recovery Timer: to recover stall of %s on route from %s, recovery count = %d

Error, filter '%s' contains an illegal type substitution character, only %s is allowed

Rendezvous Certified Advisory: %s

LDAP response resulting from checking if an entry is a member of a dynamic group:'

ignoring route '%s' at '%s', route user does not exist.

Created %s transport '%s'

Send recover request for routed queue flow stall for queue %s

Removing routed topic consumer '%s'

License has been activated.

Hostname: %s

Evaluation Software Notice: remaining uptime is %d hours %d minutes.

[%s@%s]: rejected connect from route: implicit route already exists

LDAP response resulting from getting attributes for group '%s':

ldap_parse_reference: %s

Storage Location: '%s'.

Search reference: %s

Route Recover Interval is %u seconds.

Route Recover Minimum Message Count is %u.

Route connect error: route has no zone setting

SS: Deleting existing GMD file.

LDAP error: %s

Clean all flow stalls for route to server %s: %s

%s: shutdown server

Reading configuration from '%s'.

Configuration warning: file=%s, line=%d: illegal to specify both '%s' and '%s', ignoring '%s'

Recovered flow stalled consumer for destination: %s:%s

%s: revoked all %s permissions on %s '%s'

Error sending routing information to '%s'.

Send recover request

Skipping recover request, message count % PRINTF_LL_FMT d greater than recover count

Lazy Dels: max=%d avg=%.2f min=%d

Release Holds: max=%d avg=%.2f min=%d

%s: created rvcmlistener transport '%s' name '%s' dest '%s'

ERROR: file=%s, line=%d: %s is too long.

Route '%s' connecting to url '%s'.

Route '%s' connected to url '%s' with zone '%s:%s'.

Detected tie during route creation '%s'. Resolution, keep existing route (connID=% PRINTF_LL_FMT d).

Detected tie during route creation '%s'. Resolution, destroy existing route (connID=% PRINTF_LL_FMT d).

Found connID=% PRINTF_LL_FMT d for existing route '%s', but no connection.

[%s@%s]: rejected connect from route: %s

Configuration warning: file=%s, line=%d: Use of Rendezvous Bridge via tibrv... parameters has been deprecated. This feature is subject to removal in the next release of this product. Please convert your configuration to utilize transports defined in transports.conf configuration file.

Rendezvous %s %s enabled (RV %s).

Error in ldap_search_ext_s: %s

Server is re-entering standby mode.

Statistics database memory now below limit

SS: Destroying SmartSockets transport %s

Created file '%s'

Restored routed topic consumer for '%s'

Adding routed topic consumer for '%s'

Subscriber %s for topic '%s' exceeded topic limit.

Refrained from removing configured durable '%s'

Sync Recs: max=%d avg=%.2f min=%d

SS: Unsubscribe from '%s' tport = %s

Recovered %d pending connection%s.

SS: Imported message on tport='%s', subject='%s', reply='%s'.

Clean flow stall for consumers of destination %s:%s

ldap_search_s(%x, %s, %s, %s, [NULL])

%s:%s queue browser failed: illegal to use wildcard queue [%s]

There should be only one consumer reaching %s, but %d found

%s:%s queue browser failed: cannot browse [%s] because it is a routed queue.

Clear (Non-IO) flow stalled on dest %s:%s from route of %s

Error sending routing information to %s, send failed

%s: %s updated: '%s'

%s: consumed_msg_hold_time updated: '%d'

Authorization is disabled.

SSL connect: using certificate username '%s'.

SSL reset to TCP for connID=% PRINTF_LL_FMT d, user='%s'

Configuration warning: file=%s, line=%d: invalid trace option '%s' is ignored

Server is now active.

(NON-IO) Flow stalled on dest %s from route of %s

Dump of user cache:

Administrator group not found, created with default member.

Received exception on route '%s': '%s'

ldap_search_s(%x, %s, %s, %s, [%s,%s,%s,%s,%s, NULL])

EXPIRE: msgsz=%d

Clean flow stall for routed consumer of queue %s

EXPIRE: total=% PRINTF_LLFMT d expire=0

EXITING

Configuration error: file=%s, line=%d: ignoring invalid selector specifications in route parameters

%s: created group '%s'

set %s:% PRINTF_LLFMT d in flow stall recover request

Error: unable to bind to LDAP server as: '%s', %s

DISK IO stats for %s:

Authorization exception creating routed topic consumer for '%s'

Fault tolerant reconnect timeout is set to a negative or 0 value of %d seconds,

Licensed server is waiting for license activation.

LDAP message resulting from checking existence:

Memory limit of %d MB exceeded.

%s %s to %s: connID=% PRINTF_LLFMT d consID=% PRINTF_LLFMT d
msgID='%s' %s='%s'%s%s%s%s%s%s

User '%s' is authenticated via LDAP

User '%s' is authenticated via JAAS

Route '%s' accepted from host '%s' with zone '%s:%s'.

%s:%s queue browser failed: queue does not exist: [%s]

%s acknowledged by %s: connID=% PRINTF_LLFMT d consID=%
PRINTF_LLFMT d msgID='%s' %s='%s'

%s premature exit: %s : connID=% PRINTF_LLFMT d prodID=%
PRINTF_LLFMT d msgID='%s' %s='%s'

%s: removed user '%s' from group '%s'

Reading SS configuration from '%s'.

Ignoring inbound routed topic '%s', illegal topic.

%s: Compacting %s with no time limit.

STARTING POP WAITING

Flow stall recover ack received Post IO

RVCM name not specified for transport '%s', using RVCM name '%s'

Purged % PRINTF_LLFMT d queue message%s.

Purged % PRINTF_LLFMT d topic consumer message%s.

No memory to process incoming data from connection id=% PRINTF_LLFMT d.
Connection terminated.

%s: Disconnected, connection id=% PRINTF_LLFMT d, reason: %s%s

%s: connection id=% PRINTF_LLFMT d purged after FT timeout

Error, missing %s parameter

Bytes: max=%d avg=%.2f min=%d

Server is active.

%s: %s bridge: source=[%s:%s] target=[%s:%s]

Error, filter '%s' contains too many occurrences of %%%s, max allowed is: %d

%s: created JNDI name '%s' for %s '%s'

Server is in standby mode.

%s: create %s failed: durable access denied for %s [%s].

%s: Destroyed producer (connid=% PRINTF_LLFMT d, sessid=%
PRINTF_LLFMT d, prodid=% PRINTF_LLFMT d) %s

ldap_simple_bind_s(, *****)

[%s] %s

Active server '%s' not found.

Backup server '%s' has connected.

Error in ldap_set_option: %s

%s:%s queue browser failed: access denied for queue [%s]

Ignoring inbound routed topic '%s', no corresponding topic.

%s: created user '%s'

Unable to initialize route: expected route name '%s', received '%s'.

Evaluation Software Notice: remaining uptime is %d minutes.

%s: created topic '%s'%s%s

Connected to LDAP server %s

Processed %d msgs

Error, LDAP is disabled

Configuration warning: file=%s, line=%d: illegal to use '.' in server name, replaced with '_',

%s: %s %s '%s' administrative permissions: %s

Warning: statistics database memory exceeded limit

[%s@%s]: rejected connect from route: this shouldn't happen: route exists with no zone setting

Rejected connect from route '%s' at %s, routing disabled.

Missing heartbeats from primary server '%s'.

Flow stall recovery request received, send to IO

Unable to initialize route '%s': route server returned: '%s'

RUNNING SWAPPER %d %d needed = %u!

Restoring consumer warning: zone of id %d does not exist in zone mapping entries

Ignoring inbound routed queue '%s', no corresponding queue.

%s:%s queue browser failed: illegal to use reserved queue [%s]

%s: committed transaction %s

Trying to send flow stall recovery request for destination: %s

%s: destroyed connection % PRINTF_LLFMT d

ldap_search_s(%x, %s, %s, %s, [%s,%s,%s,NULL])

Allocating storage to minimum %s for store '%s', please wait.

%s:%s queue browser failed: invalid name of queue [%s]

%s: updated group '%s'

SS: Created subscriber to '%s' LB override=%d. tport='%s'

%s: destroyed message '%s'

Configuration warning: file=%s, line=%d: Use of Rendezvous import/export settings via tibrv_... parameters has been deprecated. This feature is subject to removal in the next release of this product. Please convert your configuration to utilize 'import' and 'export' properties and using transports defined in transports.conf configuration file.

Route '%s' disconnected, connection id=% PRINTF_LL_FMT d

Routed Queue '%s' is not a home Queue

Logging into file '%s'

%s: create %s failed: access denied for %s [%s].

Unable to obtain message type number for imported SS message

Route Warning: host of this name does not exist: %s

%s: created queue '%s'%s%s

now timer fired

Clock sync timer created with interval %d seconds

Clock sync timer fired

Clock sync timer error: %s

Breaking from remove thread for wantsLock!

%s: created JNDI name '%s' = '%s'

Metadata storage: '%s'.

Flow stall recover ack received for destination %s

Server rereading configuration.

Route '%s' sent resume request

Refrained from deleting configured durable '%s' even though application's attributes differ from configuration

EXPIRE: msgs=% PRINTF_LL_FMT d exp=% PRINTF_LL_FMT d expd=% PRINTF_LL_FMT d int=% PRINTF_LL_FMT d tm=% PRINTF_LL_FMT d

Server of version %d.%d does not support flow stall recovery, do nothing.

%s: rotated log file.

Asynchronous storage: '%s'.

Created Routed Dynamic Queue '%s' from '%s'

Results of searching for dynamic groups:

Transaction for non-existent consumer: % PRINTF_LL_FMT d connID=%
PRINTF_LL_FMT d sessID=% PRINTF_LL_FMT d %s

Error in ldap_unbind: %s

Rendezvous %s %s enabled.

[%s@%s]: route connect failed: invalid password

%s: updated topic '%s': %s

ldap_search_s(%x, %s, %s, %s, [%s, NULL])

Configuration warning: file=%s, line=%d: invalid option '%s' is ignored

Server is in standby mode for '%s'.

Error, references not supported

Acknowledging the flow stall recover request for destination %s:%s and resume the flow

Failed to rename file, original file %s saved as file %s. Please rename the file

Route connect error: can not connect to route '%s' at '%s', error=%s.

Recovered %d message%s.

%s: deleted group '%s'

Start opening sync db

Start opening async db

Removed Routed Dynamic Queue '%s'

%s: %s bridge: source=[%s:%s] target=[%s:%s] selector='%s'

Error, zero entries returned from getting attributes for group '%s':

Warning: configuration file 'tibjmsd.conf' should be renamed to 'tibemsd.conf'.

Warning: [queue: %s]: dynamic routed queues are not supported.

Transaction for non-existent message: % PRINTF_LL_FMT d connID=%
PRINTF_LL_FMT d sessID=% PRINTF_LL_FMT d %s

Route recovery of destination %s on route from %s failed, will try again: %s

[%s@%s]: route connect failed: route server not authorized.

Implicit route to [%s] already exists

%s: %s route '%s', URL=[%s]

Results of searching for static groups:

%s: Queue limit exceeded for queue '%s'.

%s: Topic limit exceeded for topic '%s'.

Implicit route to '%s' already exists.

%s: deleted rvcmlistener transport '%s' name '%s' dest '%s'

Evaluation Software Notice: remaining uptime is %d hours.

Secure Socket Layer is enabled, using %s

Using cryptographic module %s

Reserve memory reestablished, all client requests accepted. Pending msg count =
% PRINTF_LLFMT d

Msg recs processed = %d

ldap_search_s(%x, %s, %s, %s, [%s,%s,NULL])

(IO) Flow stalled on dest %s:%s from route of %s

Invalid specifications for route '%s' topic '%s'

%s: Created producer (connid=% PRINTF_LLFMT d, sessid=% PRINTF_LLFMT
d, prodid=% PRINTF_LLFMT d) %s

SS: Consumer subscribe to '%s' LB override=%d. tport='%s'

Routing is enabled.

Recovering state, please wait.

Files opened.

Starting msgPass.

Finished msgPass.

Administrator user not found, created with default password.

Route '%s' sent suspend request

%d batches, %.2f batches/sec

%s: purged queue '%s'

Error in ldap_search_s: %s

%s: created durable '%s' Selector: %s

Accepted license with limits: conns=%d hosts=%d hours=%d

USING %d memory

Continuing as active server.

VALIDATING STORE %s

[%s@%s]: rejected connect from route: route already connected

%s: purged topic '%s'

ldap_search_ext: %s

Flow control %s on %s '%s'

Accepting connections on %s.

Configuration warning: ignoring tibrvcn_import property set on %s '%s' because it collides with tibrvcn_import property on %s '%s'

Warning: configuration file '%s' not found and has been created. All configuration settings have been reset to defaults.

EXPIRE ERROR: oldCount=% PRINTF_LLFMT d found=% PRINTF_LLFMT d walked=% PRINTF_LLFMT d

Shutdown complete.

Restarting now.

%s: Created %s%sconsumer%s%s%s%s (connid=% PRINTF_LLFMT d, sessid=% PRINTF_LLFMT d, consid=% PRINTF_LLFMT d) on %s '%s'%s%s%s%s

Created shared %ssubscription '%s' (id=% PRINTF_LLFMT d) on topic [%s]%s%s%s%s

Search completed successfully. Entries found: %d

Created routed dynamic queue '%s' for '%s'

%s: deleted durable '%s'

Part of the DN that matches an existing entry: %s

Purged %d connection%s.

Ignoring inbound routed topic '%s', local topic is not global.

%s: deleted user '%s'

%s: create %s failed: access denied for monitoring %s [%s].

Administrator group found with no members, added default member.

%s: updated user '%s'

SmartSockets transports are enabled.

Running in Temporary Destination Compliance mode.

ldap_search_s(%x, %s, %d, %s, [%s, NULL])

%s: %screated dynamic %s '%s'

Using %d threads for LDAP processing.

Routing is disabled.

%s: %s factory '%s'

INIT-EXPIRE: exp=%d mexp=% PRINTF_LLFMT d oldt=% PRINTF_LLFMT d
interval=%d

Error in ldap_initialize(%s)

Created dynamic %s '%s'

%s: Compacting %s with time limit %PRINTF_LLFMTd seconds

Error sending route query to '%s'.

%s: updated queue '%s': %s

Server name: '%s'.

Route configuration error: global queue '%s' from route '%s' collides, global
queues must be unique.

Route configuration error: routed queue '%s' from route '%s' is not global in '%s'.

Server shutting down.

Server preparing to restart.

Route configuration warning: global queue '%s' from route '%s' not configured on
local server

Flow stall recovery request received, recover consumer of id % PRINTF_LLFMT d

Flow Control is enabled.

%s: deleted JNDI name '%s'

Clear (IO) flow stalled on dest %s:%s from route of %s

Removing route '%s', URL='%s': this route is duplicate, creates a loop or has
configuration errors

EXPIRE: giving up amid lock

Done opening async db

Error, invalid search scope: %s

Error in ldap_url_parse, returned: %d

%s: create %s failed: durable recreation access denied for %s [%s].

Ignoring inbound routed queue '%s', illegal queue.

key: '%s' value: '%s'

Connection to primary server '%s' has been lost.

Route connect error: failed connect to server '%s' at '%s'

%s: rolled back transaction %s

LDAP response resulting from checking existence:

ldap_parse_result: %s

Hostname IP address: %s

Flow stall recovery request received, after IO

%s: Connected, connection id=% PRINTF_LLFFMT d, type: %s%s

%s: Reconnected, connection id=% PRINTF_LLFFMT d, type: %s%s

Unable to initialize one hop route: One-hop routing is not supported for server version %d.%d

Error, must provide static and/or dynamic group name attribute

Unable to initialize route, expected route name not received.

Stat: rate=%d cleanup=%d memory=%d

%s: Destroyed %sconsumer%s%s%s%s (connid=% PRINTF_LLFFMT d, sessid=% PRINTF_LLFFMT d, consid=% PRINTF_LLFFMT d) on %s %s'

Destroyed shared %ssubscription '%s' (id=% PRINTF_LLFFMT d) on topic [%s]

%s: Unsubscribed durable consumer '%s' due to administrator deleting durable (consider=% PRINTF_LLFFMT d) on topic '%s'

%s: Unsubscribed shared durable subscription '%s' due to administrator deleting durable (id=% PRINTF_LLFFMT d) on topic '%s'

%s: Unsubscribed durable consumer '%s' due to user calling unsubscribe (connid=% PRINTF_LLFFMT d, consid=% PRINTF_LLFFMT d) on topic '%s'

%s: Unsubscribed shared durable subscription '%s' due to user calling unsubscribe (connid=% PRINTF_LLFFMT d, id=% PRINTF_LLFFMT d) on topic '%s'

%s %s from %s: connID=% PRINTF_LLFFMT d prodID=% PRINTF_LLFFMT d msgID='%s' %s mode=%s size=%d %s='%s'%s%s

ldap_search_s(%x, %s, %s, %s, [%s,%s,%s,%s, NULL])

ldap_search_s: %s

Error, filter '%s' too long, max length is %d characters

%s: deleted %s %s'

Disallowing Rendezvous Certified Message listener '%s'

SS: Exporting EMS message: tport='%s', dest='%s', reply='%s' SSDelivery=%d, LB=%d

Refrained from removing configured route durable '%s'

Authorization is enabled.

Rendezvous Advisory: %s

Refrained from removing durable for route '%s' due to configured durable

Configuration error: file=%s, line=%d: invalid value of option '%s'. Unable to start.

%s: %sconnect failed: %s%s

unable to create connection with existing ID % PRINTF_LLFMT d

Synchronous storage: '%s'.

Clean all flow stalls for destination %s

Done opening sync db

%s: added user '%s' to group '%s'

ldap_search_s(%x, %s, %s, %s, [%s,%s,%s,%s,%s,%s, NULL])

Configuration warning: file=%s, line=%d: can not specify 'NONE' with other options

%s: %s failed: access denied for %s [%s].

Process started from '%s'.

Clean flow stall for routed consumer of queue %s: no other remote consumers, remove the stall

[%s@%s]: connect failed: reached maximum number of %s %d

Refrained from removing durable for route '%s' due to configured durable

LDAP Cache: User '%s' is a member of group(s):

LDAP Cache: Deleting cached record of user: '%s'

Client ID is too long.

Durable name is too long.

Durable name must be specified (connID=% PRINTF_LLFMT d)

Consumed Msg Hold Time is %d seconds.

A duplicate durable instance (conn=%s durable=%s dest=%s) was detected, discarding old one

The shared subscription [%s] (id=% PRINTF_LLFMT d) already exists on topic [%s]%s%s%s%s, rejecting the new one on [%s]%s%s%s%s

%s: create durable subscription failed: detected an existing shared durable subscription [%s] (id=% PRINTF_LL_FMT d) for topic [%s]

%s: create shared durable subscription failed: detected an existing unshared durable subscription [%s] (consid=% PRINTF_LL_FMT d) for topic [%s]

%s: create shared durable subscription [%s] on topic [%s] failed: detected an existing shared durable subscription (id=% PRINTF_LL_FMT d) for topic [%s] with different attributes and active consumers

%s: create shared durable subscription [%s] on topic [%s] failed: %d - %s

A duplicate connection with same client id (clientid=%s) detected, destroying old conn (conn-id=% PRINTF_LL_FMT d)

Warning, deleting and recreating durable '%s' due to change in client attributes: %s%s%s

Unable to build monitor message. Error code: %d - %s

Multicast is enabled.

%s: Multicast consumers require NO_ACKNOWLEDGE sessions: %s

%s: Multicast consumers require non-transacted sessions: %s

%s: Multicast consumer status: %s (consid=% PRINTF_LL_FMT d)

%s: Multicast consumer status: %s (consid=% PRINTF_LL_FMT d channel='%s')

[%s]: tx=% PRINTF_LL_FMT d bytes, rtx=% PRINTF_LL_FMT d bytes, buf=% PRINTF_LL_FMT d bytes

[%s@%s %s]: rcv=% PRINTF_LL_FMT d bytes, lost=% PRINTF_LL_FMT d, naks=% PRINTF_LL_FMT d, failed=% PRINTF_LL_FMT d

%s

%s: JMX stats for store '%s' updated: %s

Route configuration: Adding topic '%s' for server %s

Route configuration: Sending %d topics to server %s at %s - %s

Route configuration: Sending topics to server %s at %s - %s

Route configuration: Sending queue '%s' to server %s at %s - %s

Route configuration: Sending queues to server %s at %s - %s

Route configuration: Processing topics from server %s at %s.

Route configuration: Processing queue '%s' from server %s at %s.

Route '%s' acknowledgment timer destroyed.

Route '%s' acknowledgment timer unknown event type.

Route '%s' acknowledgment timer send failed: %s

Route '%s' acknowledgment timer connection % PRINTF_LL_FMT d not found.

Route configuration: Processing queues from server %s at %s.

Discarded incoming client message exceeding size limit: connID=%
PRINTF_LL_FMT d, %s='%s', size=%d.

%s selector exceeded selector_logical_operator_limit of %d.

Configuration update failure: %s

Rolling back to configuration on disk.

Rollback failed: %s

Rollback succeeded.

%s property [%s].

%s configuration item:

%s status: %s

Illegal property get in _emsdConfigurationObjectProperty_GetStringValue.

Attempting to generate a search key for an unsearchable object (%s).

More than one result found searching route selectors route=%s topic=%s.

More than one result found searching alias %s=%s jndiName=%s.

Fault tolerant configurations should have a primary listens marked as ft_active -
FT will be disabled.

Fault tolerant configurations should have a secondary listens marked as ft_active
- FT will be disabled.

Configured as fault tolerant secondary.

Configured as fault tolerant primary.

Unable to find group %s.

Detected Mixed mode configuration: Ignoring property %s file=%s, line=%d

Missing field %s from server object %s near line %d.

Invalid field detected in server object (%s), field (%s), at line %d.

%s will not complete until the server is restarted.

Detected unsupported password hash for user \q%s\q. The user's password may
need to be reset.

Ignoring request to remove an administrative user.

Ignoring request to remove the administrative group.
%s will not occur until fault tolerant failover or restart.
Ignoring unknown property %s.
Configuration error: file=%s, line=%d: ignoring rvcmlistener
Illegal configuration namespace set, %s=%s Expecting %s.
Failed to write file %s: (%s)
Error loading configuration: %s
Applying configuration changes.
Configuration update status: %s
Rollback unable to load file %s.
Initialized queue browser on '%s'%s%s%s
Closed queue browser on '%s'
Creating store '%s' file '%s' ...
Converting %s format from %s to %s
First scan on '%s' is finished
Destination cursor id % PRINTF_LL_FMT d %s slot %d
Client browsing mstore-based queue %s needs to be upgraded for optimum performance.
Warning: file=%s, line=%d: multicast_udp_encapsulation is no longer a supported parameter.
Network IO thread: %d bound to Processor id: %d
Storage thread for store '%s' bound to Processor id: %d
Disconnecting connection connID=% PRINTF_LL_FMT d, because %s (consid=% PRINTF_LL_FMT d) has reached the limit of non-acknowledged messages

Category	Basic initialization failed
Description	tibemsd was unable to start.
Resolution	Correct the configuration or startup parameters and restart.
Errors	Unable to add admin user into admin group: error=(%d) %s Fault tolerant activation has to be greater than 2x heartbeat

Server heartbeat client should be non-zero and no more than a third of the client timeout server connection

Server heartbeat server should be non-zero and no more than a third of the server timeout server connection

Client heartbeat server should be non-zero and no more than a third of the server timeout client connection

Fault Tolerant configuration error, can't create loop.

Fault tolerant connection failed, fault tolerant mode not supported on '%s'.

Fault tolerant heartbeat has to be greater than 0

Initialization failed due to errors in configuration.

Initialization failed due to errors in SSL.

Initialization failed due to errors with transports.

Initialization failed. Exiting.

Initialization has failed. Exiting.

Initialization of thread pool failed (%s). Exiting.

Startup aborted.

Server failed to read configuration.

Initialization failed: storage for '%s' not found.

Failure initializing storage thread: %s.

Initialization failed due to errors with multicast.

Configuration error: dbstore_driver_name for store [%s] cannot be empty

Configuration error: dbstore_driver_url for store [%s] cannot be empty

Configuration error: dbstore_driver_dialect for store [%s] cannot be empty

Configuration error: dbstore_driver_username for store [%s] must be specified

Configuration error: dbstore_driver_password for store [%s] must be specified

Error Loading JVM: %s

Unknown Error Loading JVM

Trying JVM location: %s

Error Loading JVM: %s

Unknown Error Loading JVM

\$sys.meta store's type must be 'file' or 'dbstore'.

Configuration error: file=%s, line=%d: The parameter '%s' is not supported on this platform

Configuration error: file=%s, line=%d: The processor ID '%d' is greater than '%d', which is the maximum processor ID for this machine

Unable to bind network IO thread: %d to Processor Id: %d. Exiting!

Unable to bind storage thread for store '%s' to Processor Id: %d. Exiting!

Category	Commit failed due to prior failure or after fault-tolerant switch
Description	A warning message indicating that the commit of a client application's transaction failed either because there were earlier errors when processing this transaction or because the transaction was started on the primary server prior to a fault-tolerant failover.
Resolution	The client application should retry the transaction.
Errors	Commit failed due to prior failure or after fault-tolerant switch.

Category	Compaction failed
Description	Compaction of the store file failed.
Resolution	The most likely cause of this error is running out of memory. Shut down tibemspd and see remedies for Out of memory.
Errors	Compaction failed on file '%s': %d (%s). Please shutdown and restart tibemspd. Compaction failed on file '%s': %d (%s). Initialization of file_destination_defrag feature failed for queue '%s' (store '%s') due to an out of memory condition. Feature is disabled. file_destination_defrag of queue '%s' (store '%s') failed: %d (%s).

Category	Configured durable differs from stored one
Description	The durables configuration file specifies a durable with a given name and client identifier with attributes that are different from the identically named durable found in the meta.db file.
Resolution	Correct the durables configuration file to match the durable defined in the meta.db file or administratively delete the durable and re-define it.

Errors	Configured durable '%s' differs from durable in storage, storage version used.
Category	Create of global routed topic failed: not allowed to create dynamic topic
Description	A server received an interest notification from another server that does not match the allowed topics in its configuration.
Resolution	This only is printed when the trace includes ROUTE_DEBUG. If the server's topic definitions are as expected, this statement can be ignored or remove the ROUTE_DEBUG trace specification to prevent printing.
Errors	Create of global routed topic failed: not allowed to create dynamic topic [%s].
Category	Create of routed queue failed: not allowed to create dynamic queue
Description	A warning indicating that a tibemsd with a route to this daemon has a queue configured to be global but this daemon does not permit the creation of that queue dynamically.
Resolution	Add the specified queue or a pattern that includes it to this daemon if you want the queue to be accessible from this daemon, otherwise the warning can be ignored.
Errors	Create of routed queue failed: not allowed to create dynamic queue [%s].
Category	Database record damaged
Description	An error occurred reading one of the tibemsd store files.
Resolution	Send details of the error and the situation in which it occurred to TIBCO Support.
Errors	Server failed to recover state.
Category	Duplicate message detected
Description	Warning generated when tibemsd receives a message with a message id that matches another message's message id.
Resolution	Only seen when message id tracking is enabled.
Errors	Detected duplicate %s message, messageID='%s'

Category	Dynamic Module Loading Errors
Description	An error occurred when loading or using a shared library module.
Resolution	Module loading is affected by the presence of shared libraries in the module path. Use the +load tracing flag to get more information about how the server is loading modules. See the section on Starting the EMS Server for more details.
Errors	<p>Problem loading %s: %s</p> <p>Unknown problem loading %s.</p> <p>Loaded %s</p> <p>Problem binding %s: %s</p> <p>Unknown problem binding %s.</p> <p>Unable to locate %s</p> <p>Fatal error: Returned from exec(), errno = %d</p> <p>OpenSSL library version mismatch</p>

Category	Error in configuration file
Description	The server encountered an invalid configuration statement in the specified configuration file on the specified line.
Resolution	Examine the appropriate configuration file and correct the syntax error.
Errors	<p>Configuration warning: file=%s, line=%d: route '%s' does not have a user configured for authorization.</p> <p>SSL Configuration error: file=%s, line=%d: invalid certificate file name, unknown extension or invalid encoding specification</p> <p>Configuration error: file=%s, line=%d: illegal to specify %s for routed queue</p> <p>Configuration error: file=%s, line=%d: bad destination specification: %s</p> <p>Configuration warning: file=%s, line=%d: illegal to specify prefetch=none for global or routed queue. Prefetch reset to default.</p> <p>Configuration warning: file=%s, line=%d: illegal to specify prefetch=none for topic. Prefetch reset to default.</p> <p>Configuration error: file=%s, line=%d: ignored alias '%s' for %s '%s' because such alias already exists</p>

Configuration error: The specified file '%s' is empty or does not exist

Configuration error: file=%s, line=%d: both tibrv_export and tibrvcm_export are specified, ignoring tibrv_export

Configuration error: file=%s, line=%d: ignoring transport '%s' in %s list, transport not found

Configuration error: file=%s, line=%d: multiple bridge entries for the same destination '%s' are not allowed.

Configuration error: file=%s, line=%d: Ignoring durable, name cannot start with \$sys.route, use route property instead.

Configuration error: file=%s, line=%d: Rendezvous transport not specified for Rendezvous CM transport '%s'

Configuration error: file=%s, line=%d: ignoring invalid max connections in the line, reset to unlimited

Configuration error: file=%s, line=%d: ignoring invalid max_client_msg_size in the line, reset to unlimited

Configuration error: file=%s, line=%d: value of %s out of range, reset to default

Configuration error: max_msg_field_print_size >= max_msg_print_size, resetting both to default

Configuration error: file=%s, line=%d: unable to create %s '%s': invalid destination name, invalid parameters or out of memory

Configuration error: file=%s, line=%d: value of db_pool_size too big or less than allowed minimum, reset to default value of %d bytes

Configuration error: file=%s, line=%d: Ignoring durable, route does not allow clientid, selector or nlocal.

Configuration error: file=%s, line=%d: Route '%s' does not exist for configured durable.

Configuration error: file=%s, line=%d: unable to process selector in route parameters, error=%s

Configuration error: file=%s, line=%d: both tibrv_import and tibrvcm_import are specified, ignoring tibrv_import

Configuration error: file=%s, line=%d: ignored route '%s' because route represents route to this server.

Configuration error: file=%s, line=%d: ignoring invalid topic selector specifications in route parameters

Configuration error: file=%s, line=%d: value of max_msg_memory less than allowed, reset to %dMB

Configuration error: file=%s, line=%d: ignored alias '%s' for factory because such alias already exists

Configuration error: file=%s, line=%d: invalid certificate file name, unknown extension or invalid encoding specification

Configuration error: file=%s, line=%d: ignored route '%s' because route has invalid zone information.

Configuration error: file=%s, line=%d: ignored route '%s' because route with such name or URL already exists.

Configuration error: file=%s, line=%d: value of msg_pool_size invalid or too big or less than allowed minimum of %d, reset to default value of %d

SSL Configuration error: file=%s, line=%d: invalid private key file name, unknown extension or invalid encoding specification

Configuration conflict: file=%s, line=%d: value of msg_pool_block_size already set at line=%d. Ignoring msg_pool_size.

Configuration error: file=%s, line=%d: bridge has no targets, unable to process

Configuration error: file=%s, line=%d: Illegal to specify routed queue as a bridge source

Configuration error: file=%s, line=%d: client_trace error: %s

Configuration error: file=%s, line=%d: %s

Configuration error: file=%s, line=%d: duplicate specification of transport type

Configuration error: file=%s, line=%d: duplicate value

Configuration error: file=%s, line=%d: Ignoring durable, duplicate of earlier entry.

Configuration error: file=%s, line=%d: Ignoring durable, name is invalid.

Configuration error: file=%s, line=%d: Ignoring durable, name is missing or invalid.

Configuration error: file=%s, line=%d: Ignoring durable, topic is invalid.

Configuration error: file=%s, line=%d: Ignoring durable, topic is missing or invalid.

Configuration error: file=%s, line=%d: error in the bridge description, unable to proceed.

Configuration error: file=%s, line=%d: error in permissions

Configuration error: file=%s, line=%d: error in the transport description, unable to proceed.

Configuration error: file=%s, line=%d: errors in line, some options may have been ignored

Error: unable to add bridge specified in file=%s, line=%d. Error=%s

Configuration error: file=%s, line=%d: Unable to create destination defined by the bridge source

Unable to create Rendezvous Certified transport '%s' because it references undefined Rendezvous transport '%s'

Configuration error: file=%s, line=%d: failed to create ACL entry, reason=%s

Unable to export message to SmartSockets. error=%s.

Use fsync error: file=%s, line=%d: invalid property value

Use fsync (min disk) error: file=%s, line=%d: invalid property value

exit_on_nonretryable_disk_error: file=%s, line=%d: invalid boolean property value

consumed_msg_hold_time: file=%s, line=%d: invalid property value

active_route_connect_time: file=%s, line=%d: invalid property value

Fault tolerant reread error: file=%s, line=%d: invalid property value

Fault standby lock check error: file=%s, line=%d: invalid property value

Configuration error: file=%s, line=%d: ignored unknown permission '%s'

Configuration error: file=%s, line=%d: ignoring duplicate %s '%s' specified earlier

Configuration error: file=%s, line=%d: ignoring duplicate transport name '%s' in %s list

Configuration error: file=%s, line=%d: ignoring duplicate user

Configuration error: file=%s, line=%d: ignoring errors in permission line

Configuration error: file=%s, line=%d: ignoring invalid connect attempt count

Configuration error: file=%s, line=%d: ignoring invalid connect attempt delay

Configuration error: file=%s, line=%d: ignoring invalid connect attempt timeout

Configuration error: file=%s, line=%d: ignoring invalid disk statistic period

Configuration error: file=%s, line=%d: ignoring invalid entry syntax

Configuration error: file=%s, line=%d: ignoring invalid factory load balancing metric

Configuration error: file=%s, line=%d: ignoring invalid ft activation in the line

Configuration error: file=%s, line=%d: ignoring invalid ft heartbeat in the line

Configuration error: file=%s, line=%d: ignoring invalid ft reconnect timeout in the line

Configuration error: file=%s, line=%d: ignoring invalid line

Configuration error: file=%s, line=%d: ignoring invalid line in factory parameters

Configuration error: file=%s, line=%d: ignoring invalid line in route parameters

Configuration error: file=%s, line=%d: ignoring invalid line: invalid syntax in the line

Configuration error: file=%s, line=%d: ignoring invalid reconnect attempt count

Configuration error: file=%s, line=%d: ignoring invalid reconnect attempt delay

Configuration error: file=%s, line=%d: ignoring invalid reconnect attempt timeout

Configuration error: file=%s, line=%d: ignoring invalid value of %s

Configuration error: file=%s, line=%d: ignoring invalid value '%s' for property '%s'

Configuration error: file=%s, line=%d: ignoring unknown property '%s'

Configuration error: file=%s, line=%d: ignoring unrecognized property '%s'

Configuration error: file=%s, line=%d: ignoring user out of group context

Configuration error: file=%s, line=%d: illegal to use predefined name %s

Configuration error: file=%s, line=%d: Invalid clientid value

Configuration error: file=%s, line=%d: invalid value of db_pool_size, reset to default of %d bytes

Configuration error: file=%s, line=%d: invalid line syntax or line out of order

Configuration error: file=%s, line=%d: invalid value of max memory, reset to unlimited

Configuration error: file=%s, line=%d: invalid value of max_msg_memory, reset to unlimited

Configuration error: file=%s, line=%d: invalid property value

Configuration error: file=%s, line=%d: invalid property value, reset to default.

Configuration error: file=%s, line=%d: invalid password

Configuration error: file=%s, line=%d: invalid value of reserve_memory, reset to zero

Configuration error: file=%s, line=%d: invalid value of route_recover_interval, reset to default %d

Configuration error: file=%s, line=%d: invalid value of route_recover_count, line ignored

Configuration error: file=%s, line=%d: Invalid selector value

Configuration error: file=%s, line=%d: invalid syntax of %s, unable to continue.

Configuration error: file=%s, line=%d: invalid transport parameter '%s'

Configuration error: file=%s, line=%d: invalid transport type '%s'

Configuration error: file=%s, line=%d: invalid trace_client_host value

Configuration error: file=%s, line=%d: invalid trace_millisecond value

Configuration error: file=%s, line=%d: invalid value of %s, reset to unlimited

Configuration error: file=%s, line=%d: invalid value '%s'

Configuration error: file=%s, line=%d: invalid value '%s' for parameter '%s'

Configuration error: file=%s, line=%d: invalid value of '%s'

Configuration error: file=%s, line=%d: invalid value of %s

Configuration error: file=%s, line=%d: invalid value of %s, reset to 256MB

Configuration error: file=%s, line=%d: invalid value of %s, reset to default

Configuration error: file=%s, line=%d: line too long, ignoring it

Configuration error: file=%s, line=%d: maximum number of listen interfaces reached.

Configuration error: file=%s, line=%d: multiple principals specified, line ignored

Configuration error: file=%s, line=%d: multiple targets specified, line ignored

Configuration error: file=%s, line=%d: out of memory, unable to create Rendezvous transport

Configuration error: file=%s, line=%d: no permissions found in acl entry

Configuration error: file=%s, line=%d: no target found in acl entry

Configuration error: file=%s, line=%d: %s '%s' not found

Configuration error: No topic exists for configured durable '%s%s%s'.

failed to create durable '%s', exception: %s.

Configuration error: file=%s, line=%d: no valid user or group found in acl entry

Configuration conflict: file=%s, line=%d: Overriding value of msg_pool_size already set at line=%d.

Configuration warning: file=%s, line=%d: parameter '%s' is deprecated

Configuration error: file=%s, line=%d: value of reserve_memory too small, reset to 16MB

Configuration error: file=%s, line=%d: ignoring invalid line in route parameters: invalid zone type, too long

Configuration error: file=%s, line=%d: ignoring invalid line in route parameters: zone name exceeding %d bytes

Routing Configuration error: file=%s, line=%d: invalid property value

Configuration warning: file=%s, line=%d: ignoring rvcmlistener, duplicate

Configuration error: file=%s, line=%d: ignoring rvcmlistener, first token is invalid

Configuration error: file=%s, line=%d: ignoring rvcmlistener, invalid destination

Configuration error: file=%s, line=%d: ignoring rvcmlistener, second token is invalid

Configuration error: file=%s, line=%d: ignoring rvcmlistener, third token is invalid

Configuration error: file=%s, line=%d: ignoring rvcmlistener, wildcards are not permitted

SmartSockets configuration directory name is too long. error=%s.

SmartSockets file '%s' not found.

SSL Configuration error: file=%s, line=%d: duplicate value

SSL Configuration error: file=%s, line=%d: invalid value of DH key size.

SSL Configuration error: file=%s, line=%d: invalid property value

Configuration error: file=%s, line=%d: syntax error in the line, ignoring

Configuration error: file=%s, line=%d: syntax errors in line, line ignored

Topic '%s' not valid in configured durable '%s'.

%s%sNo client ID for%s unshared durable '%s'.

Configuration error: file=%s, line=%d: Unrecognized attribute

Configuration error: file=%s, line=%d: user '%s' not found, ignoring

Configuration error: file=%s, line=%d: value is invalid or less than minimum %d, reset to 0

Configuration error: file=%s, line=%d: value less than allowed minimum, reset to 0

Configuration error: file=%s, line=%d: value of %s less than allowed minimum of %dKB, reset to unlimited

Configuration error: file=%s, line=%d: Invalid value or value does not fall between %d and %d

Configuration error: Invalid line: file=%s, line=%d

Configuration error: Missing store header: file=%s, line=%d

Configuration error: Mixed mode configuration: file=%s, line=%d

Configuration error: Invalid store parameter: file=%s, line=%d

Configuration error: Store definition failed

Configuration error: Unrecognized store type requested.

Configuration error: Filename for store '%s' cannot be empty.

Error occurred writing store definition into file.

Configuration error: file=%s, line=%d: ignoring channel '%s' on topic '%s', channel does not exist

Configuration error: file=%s, line=%d: ignoring channel '%s' on topic '%s', overlaps with channel '%s' on topic '%s'

Configuration error: file=%s, line=%d: ignoring channel '%s', duplicate name

Configuration error: file=%s, line=%d: ignoring channel '%s', address of '%s:%d' already defined

Configuration error: file=%s, line=%d: channel '%s', %s

Configuration error: file=%s, line=%d: channel '%s', no address specified.

Configuration error: file=%s, line=%d: channel '%s', invalid address syntax: port not specified.

Configuration error: file=%s, line=%d: channel '%s', invalid address: group must be in the range 224.0.0.0 to 239.255.255.255

Configuration error: file=%s, line=%d: channel '%s', interface must address a valid multicast-capable network interface.

Configuration error: file=%s, line=%d: channel '%s', invalid address: port must be in the range 1 to 65535

Configuration error: file=%s, line=%d: channel '%s', ttl must be in the range 1 to 255

Configuration error: file=%s, line=%d: channel '%s', priority must be in the range -5 to 5

Configuration error: file=%s, line=%d: channel '%s', maxrate must be less than 512MB

Configuration error: file=%s, line=%d: channel '%s', maxtime must be greater than 0

Configuration error: file=%s, line=%d: cannot store messages in: %s

Configuration error: file=%s, line=%d: cannot find store: %s

Required store param 'type' not specified for store '%s'

Configuration error: file=%s, line=%d: parameter does not match another parameter that defined store '%s' as 'file' type%s.

Configuration error: file=%s, line=%d: parameter does not match another parameter that defined store '%s' as 'dbstore' type%s.

Configuration error: file=%s, line=%d: parameter does not match another parameter that defined store '%s' as 'mstore' type%s.

Store '%s' already defined

Configuration error: Store with similar dbstore_driver_url exists, file=%s, line=%d

Configuration error: duplicate file name %s for stores %s and %s

Configuration warning: file=%s, line=%d: the discardAmount is too small for the selected RV Queue Limit Policy. It is recommended to have at least 10%% of the maxEvents

Configuration error: file=%s, line=%d: the discardAmount is too big compared to the maxEvents value. Defaulting to TIBRVQUEUE_DISCARD_NONE policy

Configuration error: file=%s, line=%d: maxEvents and discardAmount values must be strictly positive for an RV Queue Limit Policy other than TIBRVQUEUE_DISCARD_NONE. Defaulting to TIBRVQUEUE_DISCARD_NONE policy

Configuration error: file=%s, line=%d: RV Queue Limit Policy '%s' unknown or not supported. Defaulting to TIBRVQUEUE_DISCARD_NONE policy

Configuration error: file=%s, line=%d: Error parsing the RV Queue Limit Policy value '%s'. Defaulting to TIBRVQUEUE_DISCARD_NONE policy

Configuration warning: file=%s, line=%d: The bridge's source destination '%s' is dynamic but has no parent. The bridge should either be removed or a static parent destination added

Category	Error writing commit request, errors already occurred in this transaction
Description	A client application's attempt to commit a transaction failed because the server encountered an error during an operation associated with the transaction.

Resolution	Examine previous error statements to determine the cause of the operation failure and correct that before attempting the transaction again.
Errors	Error writing commit request, errors already occurred in this transaction.
Category	Error writing configuration file
Description	tibemspd was unable to update one of its configuration files following a configuration change.
Resolution	Check that the user that started the tibemspd has permission to change the configuration files and that there is sufficient disk space on the device.
Errors	Error occurred saving acl information Error occurred saving bridges information Error occurred saving durables information Error occurred saving factories information Error occurred saving file '%s' Error occurred saving group information Error occurred saving %s information Error occurred saving main configuration file '%s' Error occurred saving routes information Error occurred saving tibrvcm information Error occurred while updating main configuration file '%s'. Configuration has not been saved. Error occurred writing bridges into file. Error occurred writing destination '%s' into file Error occurred writing factory into file. Error occurred writing group '%s' into file Error occurred writing into the file '%s'. Error occurred writing route into file. I/O error occurred saving bridge information I/O error occurred saving group information I/O error occurred saving route information

I/O error occurred writing into file '%s'

Category	Error writing to store file
Description	tibemsd was unable to write data to one of its store files.
Resolution	Ensure that the directory containing the store files is mounted and accessible to the tibemsd, and that there is free space available on the device
Errors	<p>Failed writing block data to '%s': %s</p> <p>Failed writing message to '%s': I/O error or out of disk space.</p> <p>Failed writing purge state for queue '%s': I/O error or out of disk space.</p> <p>Failed writing purge state for topic consumer: I/O error or out of disk space.</p> <p>Exception trying to create confirm record, %s.</p> <p>Exception trying to create message from store: %s</p> <p>Exception trying to create transaction record.</p> <p>Exception trying to create valid messages record, %s.</p> <p>Exception trying to export message to RV.</p> <p>Failed writing message to '%s': %s.</p> <p>Exception writing transaction commit record: %s.</p> <p>Exception writing transaction rollback record: %s.</p> <p>Exception writing transaction prepare record: %s.</p> <p>Failure deleting old version of transaction record: %s.</p> <p>Failed deleting '%s' record from %s: %s</p>

Category	Errors in Database Stores Setup
Description	In a database stores setup, errors occurring at runtime
Resolution	Check your database server vendor and database administrator for failures occurring during writes, deletes, reads of different records, for failures occurring during database store open check with the database administrator for permissions and the existence of the database. For failures occurring during a FT setup where all the stores are database stores, please check with the database server vendor or database administrator. In the case where both are active, we recommend shutting down both the servers and investigating the problem.

Errors Unable to open store [%s]: [ESTATUS = %d, ERRSTR = %s]

Failed to store message record in store [%s]: [ESTATUS = %d, ERRSTR = %s]

Failed to write ack record in store [%s]: [ESTATUS = %d, ERRSTR = %s]

Failed to write txn record in store [%s]: [ESTATUS = %d, ERRSTR = %s]

Failed to update txn record in store [%s]: [ESTATUS = %d, ERRSTR = %s]

No memory to create no hold list for valid msgs record

No memory to create hold list for valid msgs record

No memory to create held list for valid msgs record

Failed to write valid msg record in store [%s]: [ESTATUS = %d, ERRSTR = %s]

Failed to update msg record with record id [% PRINTF_LLFMT d] in store [%s]: [ESTATUS = %d, ERRSTR = %s]

Failed to delete %s record id = % PRINTF_LLFMT d : [ESTATUS = %d, ERRSTR = %s]

Failed to read message with store id = % PRINTF_LLFMT d: [ESTATUS = %d, ERRSTR = %s]

Failed to initialize dbstore [%s]: [ERRSTR = %s]

Failed to open store [%s], error = %s

Unable to restore %s records from store [%s]: [ESTATUS = %d, ERRSTR = %s]

Failed to delete meta record: [ESTATUS = %d, ERRSTR = %s]

Failed to beginTransaction: [ESTATUS = %d, ERRSTR = %s]

Failed to read message with store id = % PRINTF_LLFMT d: [ESTATUS = %d, ERRSTR = %s]

Store [%s] locked by server %s

Store [%s] cannot be locked by server %s

Failed to store txn record: [txn id = % PRINTF_LLFMT d, ESTATUS = %d]

Failed to update txn record: [txn record id = % PRINTF_LLFMT d, ESTATUS = %d]

Exception while processing msg from database store [%s], error = %d

Failed to write meta record: [ESTATUS = %d, ERRSTR = %s]

Failed to update meta record: [ESTATUS = %d, ERRSTR = %s]

Failed to write connection record: error = %d

Failed to write session record: error = %d

Failed to write consumer record: error = %d
 Failed to write producer record: error = %d
 Failed to write zone record: error = %d
 Failed to update connection record: error = %d
 Failed to update consumer record: error = %d
 Failed to write purge record: [ESTATUS = %d, ERRSTR = %s]
 Commit Transaction Failed [ESTATUS = %d, ERRSTR = %s]
 No Memory to create lock manager: Store [%s] cannot be locked by server %s
 Could not find system record for store [%s]

Category	Exceeded system resources.
Description	The system resources are inadequate for timely processing of server activities.
Resolution	Increase the specified resource or reduce the workload on this server.
Errors	Slow clock tick %d, delayed messaging and timeouts may occur.

Category	Failed to open TCP port
Description	tibemspd was unable to open the tcp port.
Resolution	Shutdown process that is using the port or change the value of the 'listen' parameter in the server's tibemspd.conf file to a port that is not in use.
Errors	Binding connection to TCP port %d failed:%d (%s).

Category	File access error
Description	tibemspd was unable to properly access the specified file.
Resolution	Check that the path name is correct and the directory exists, the user that started tibemspd has permission to read the specified directory and path, the file exists if it isn't one that the tibemspd can create, the file is not being used by another tibemspd or some other process.
Errors	Configuration file '%s' not found.

Failed to create file '%s'

failed to open file '%s'.

failed to open log file '%s'.

Failed to read message from store.

Failed to rename file %s into %s: %s

Unable to open metadata file '%s', error '%s'.

Unable to open metadata file '%s', file may be locked.

Unable to open store file '%s', error '%s'.

Unable to open store file '%s', file may be locked.

Unable to preallocate storage file '%s'.

I/O error occurred reading from the file '%s'.

Exiting on non-retryable disk error: %d

Exception trying to read message from store.

Error during file close of '%s' - %d.

Category	FIPS 140-2 Mode Errors
Description	An error occurred while starting or running the server in FIPS 140-2 compliant mode.
Resolution	Check the configuration of SSL related parameters to make sure that no incompatible ciphers or operations are requested.
Errors	Cannot specify ldap_tls_cipher_suite in FIPS 140-2 mode. Cannot specify ldap_tls_rand_file in FIPS 140-2 mode. Cannot specify SSL cipher list in FIPS 140-2 mode. Cannot specify random data source file in FIPS 140-2 mode. Cannot specify ssl_dh_size in FIPS 140-2 mode. Cannot specify ssl_server_ciphers in FIPS 140-2 mode. Cannot specify ssl_rand_file in FIPS 140-2 mode.

Category General Multicast Status Codes and Errors

Description	General multicast errors that can occur in the Server and Multicast Daemon.
Resolution	Check the configuration of the Multicast Daemon and Server, as well as the health of the network.
Errors	<p>PGM ERROR: %s - %s (%d)</p> <p>PGM ERROR: channel=\q%s\q - %s (%d)</p> <p>Error setting PGM parameter %s=%u: %s (%d)</p> <p>Error setting PGM parameter %s=\q%s\q: %s (%d)</p> <p>Error getting PGM parameter \q%s\q: %s (%d)</p> <p>Error getting PGM statistic \q%s\q: %s (%d)</p> <p>Received an invalid EMS Message.</p> <p>Received a message spanning multiple fragments.</p> <p>PGM Session was reset for channel \q%s\q, PGM seqno=%PRINTF_LLFMt d, code=%c</p> <p>Stopped receiving on channel \q%s\q</p> <p>Started receiving on channel \q%s\q</p> <p>Error receiving on channel \q%s\q</p> <p>Stopped sending on channel \q%s\q</p> <p>Started sending on channel \q%s\q</p> <p>Error creating sender on channel \q%s\q: %s</p>
Category	Internal error that should be status-driven
Description	The server detected an internal inconsistency.
Resolution	Send the error statement and a description of the environment to TIBCO Support.
Errors	<p>**Error** unable to process message, error = %s</p> <p>Admin user not found during initialization</p> <p>Error bridging transacted data message, '%s'.</p> <p>Error processing xa commit request, %s. connID=% PRINTF_LLFMt d %s</p> <p>Error processing xa end - transaction marked ROLLBACKONLY, %s. connID=% PRINTF_LLFMt d sessID=% PRINTF_LLFMt d %s</p>

Error processing xa prepare request, %s. connID=% PRINTF_LL_FMT d %s

Error processing xa rollback request, %s. connID=% PRINTF_LL_FMT d %s

Error decoding sequence data in xa rollback request. connID=% PRINTF_LL_FMT d %s

Error decoding sequence data in route ack response.

Unable to create internal session

Problem setting flow stall recover message on route queue: %s: %s

Failed to handle connection initialization: %s.

Problem trying to recover routed consumer for queue %s: setting recover message. Error: %s

Failed to send the flow stall recover request: %s.

Unable to handle transacted data message, '%s'.

Unable to handoff connection init message: %s.

Unable to initialize fault tolerant connection, remote server returned '%s'

Unable to process producer message, failed to add sender name, error=%s.

Unable to process sequence for message.

Unable to send recover ack on flow stall: %s

Handling of route flow stall recovery request from %s failed: unable to get message property %s: %s

Handling of route flow stall recovery request failed: Unable to get message properties: %s

Failed to send acknowledge to the stall recover request of server %s, will try later. Error: %s

failed to send recover ack on stalled flow: invalid consumer

unable to create recovered connection, status: %s

Exception creating purge record.

Exception creating zone.

Exception creating zone: adding zone to state.

Exception in startup, exiting.

Exception preparing message for client send (%s): %s

Exception sending flow recover acknowledge

Exception sending routing information to %s - %s

Exception sending session init response
 Exception sending queue acknowledge response to %s: %s
 Exception trying to initialize connection.
 Exception trying to initialize connection, can't send response: %s
 Exception trying to initialize route.
 Exception trying to initialize route '%s' configured durables: %s
 Exception trying to process message, '%s'.
 Exception trying to process message from store.
 Failure queuing incoming message for processing: %s.
 Failure queuing message for removal from system: %s.
 Failure queuing message to add to dead queue: %s.
 Failure discarding topic overflow: %s.
 Failure processing system request.
 Failure processing transaction message.
 Failure bridging incoming message: %s.
 Failure verifying uniqueness of routed message: %s.
 Failure scheduling message hold release: %s.
 Exception adding message write context: %s.
 %s: Failure processing multicast request: %s
 %s: Failure sending multicast request response: %s
 %s: Failure processing multicast status: %s
 %s: Failure sending multicast status response: %s
 %s: Failure sending multicast configuration: %s
 Failure sending multicast message on channel '%s': %s
 Failure enqueueing multicast message on channel '%s': %s
 Failure starting multicast engine: %s
 Failure starting multicast channel '%s': %s
 Failure posting multicast channel '%s' wake event: %s
 Failed preparing message for writing: %s
 Failed discarding local transaction: %s

Abandoning transaction record due to IO failure.
Error sending acknowledgment to route '%s': %s
Error processing acknowledgments from route '%s': %s
Failure starting delivery of delayed message seq = % PRINTF_LLFMtd: %s
Failure moving failed delivery delayed message seq = % PRINTF_LLFMtd to
dead queue: %s

Category	Invalid connection
Description	Warning indicating that tibemspd was attempting to reestablish delivery of messages across a route to another tibemspd but was unable to find the connection for that route.
Resolution	Either reduce the tibemspd's memory requirement by consuming messages or removing messages from its queues, or increase the amount of memory available to the tibemspd by shutting down other processes on the machine or increasing the machine's memory.
Errors	Recovery flow stall for destination %s failed: invalid route connection

Category	Invalid destination
Description	An application is attempting to use a destination name that is not valid.
Resolution	Alter application code to use an acceptable destination name.
Errors	%s: create %s failed: Not permitted to use reserved queue [%s]. %s: %s failed: illegal to use wildcard %s [%s]. %s: %s failed: %s [%s] not configured. At least one bridge is referencing %s [%s] as a target. This destination does not exist and there is no parent that would allow its dynamic creation. The destination has been forcefully created. To avoid this, the bridge(s) referencing this target should be destroyed. Use of '\$' destination prefix is not supported [%s %s].

Category	Invalid listen specification
----------	------------------------------

Description	The server could not parse the listen parameter in the tibemsd.conf file
Resolution	Correct the listen parameter to match the form [protocol]://[url] as specified in the manual.
Errors	Invalid listen specification: '%s'. Invalid request to create temporary destination.
Category	Invalid session
Description	tibemsd received a request that referred to a session that doesn't currently exist.
Resolution	Send details of the error and the situation in which it occurred to TIBCO Support.
Errors	Cannot find session for ack Cannot find session for ack range %s: destroy %s failed: invalid session id=%d. Unable to destroy session, invalid session. Invalid session in commit request. Invalid commit request. Invalid session trying to update(%d) tx record. Invalid session in commit transaction record. Invalid session in recover request. Invalid session in rollback request. Invalid session in xa end request. connID=% PRINTF_LLFMT d Invalid session in xa start request. connID=% PRINTF_LLFMT d
Category	LDAP error - should always display LDAP error
Description	An attempt to authenticate a client's userid and password using the external LDAP server failed.
Resolution	Examine the error code printed by the messaging server and consult the manual for the external LDAP server.
Errors	Filter '%s' contains an illegal type substitution character, only %%s is allowed

Filter '%s' contains too many occurrences of %%, max allowed is: %d

Filter '%s' too long, max length is %d characters

Invalid search scope: %s

LDAP Configuration error: file=%s, line=%d: invalid property value

LDAP is not present

LDAP search resulted %d hits.

ldap_url_parse failed, returned: %d

Lookup of group '%s' produced incorrect or no results

Missing LDAP URL

Missing %s parameter

Zero entries returned from getting attributes for group '%s':

Failed adding user '%s' into LDAP user cache

Category	LICENSE WARNING
Description	The server detected a violation of its license.
Resolution	This error only occurs with the evaluation version of the server or in an embedded form. To correct this error either replace your evaluation version with a production version or contact the vendor who supplied the embedded version.
Errors	License violation: %s.

Category	Missing configuration
Description	An essential attribute has not been configured.
Resolution	Change the tibemsd.conf file so that a value for the attribute is provided.
Errors	Configuration error with metadata database. Configuration error with storage databases.

Category	Missing transaction
----------	---------------------

Description	A client application attempted to change the state of a transaction that the tibemspd does not have in its list of current transactions.
Resolution	Check tibemspd trace logs to see if the transaction had been committed or rolled back by an administrator, if not then check the client code to see if it or its transaction manager are calling the transaction operations in the correct order.
Errors	<p>Cannot find transaction referred to transaction record update(%d) request. connID=% PRINTF_LL_FMT d %s</p> <p>Cannot find transaction referred to in xa commit request. connID=% PRINTF_LL_FMT d %s</p> <p>Cannot find transaction referred to in xa prepare request. connID=% PRINTF_LL_FMT d %s</p> <p>Cannot find transaction referred to in xa rollback request. connID=% PRINTF_LL_FMT d %s</p> <p>Received prepare request for transaction already prepared. connID=% PRINTF_LL_FMT d %s</p> <p>Cannot find transaction referred to in xa start (resume) request. connID=% PRINTF_LL_FMT d sessID=% PRINTF_LL_FMT d %s</p> <p>Cannot find transaction referred to in xa start (join) request. connID=% PRINTF_LL_FMT d sessID=% PRINTF_LL_FMT d %s</p> <p>Cannot find transaction referred to in xa end request. connID=% PRINTF_LL_FMT d sessID=% PRINTF_LL_FMT d %s</p>
Category	Mstore errors
Description	An error occurred using an Mstore database file.
Errors	<p>Unable to open store %s: %s: %d (%s).</p> <p>Wrong schema version. Found %d, expected %d.</p> <p>Schema creation failed: '%s' error: %d %s</p> <p>Unable to reset a statement (%s): %s.</p> <p>Unable to step a statement (%s): %s: %d.</p> <p>Store %s: %s: bind fail: %d.</p> <p>Store %s: Fail retrieving consumer interest: %d.</p> <p>Store %s: Fail retrieving msg interest info: %s.</p>

Store %s: Fail writing transaction record: %s.

Store %s: Fail reading data: %s.

Store %s: Fail reading topic message: %s.

Store %s: Fail marking topic message non-pending: %s.

Store %s: Fail reading next topic message: %s.

Store %s: Fail reading queue message: %s.

Store %s: Fail getting next queue message: %s.

Store %s: Fail marking queue message non-pending: %s.

Store %s: Fail writing transaction info: %s.

Store %s: Fail deleting transaction acks: %s.

Store %s: Fail recording transaction msg: %s.

Store %s: Fail recording transaction acks: %s.

Store %s: Fail deleting ack: %s.

Store %s: Fail completing transaction: %s.

Store %s: Too many entries in memory message interest.

Store %s: Invalid message interest for destination % PRINTF_LLFMT d.

Store %s: Invalid destination read.

Store %s: Failure restoring transaction: %s.

Store %s: Failure restoring transaction msg: %s.

Store %s: Failure restoring transaction ack: %s.

Store %s: Failure resetting topic: %s.

Store %s: Correct functioning cannot be guaranteed due to mstore failure.
Exiting.

Failed writing to mstore: I/O error or out of disk space.

Category	Multicast channel allotted bandwidth exceeded.
Description	Indicates that a multicast channel's allotted bandwidth has been exceeded.
Resolution	Either slow down the publisher(s), enable flow control, or increase the multicast channel's allotted bandwidth by increasing the channel's maxrate property or increasing the server's multicast_reserved_rate property.

Errors	Multicast channel \q%s\q has exceeded its allotted bandwidth
Category	Multicast Daemon Status Codes and Errors
Description	Errors occurring in the Multicast Daemon.
Resolution	Check the configuration of the Multicast Daemon and Server, as well as the health of the network.
Errors	<p>Interface IP address: %s</p> <p>[%s] Connection created, connid=% PRINTF_LLFMT d</p> <p>Error: Unable to set channel property \q%s\q=% PRINTF_LLFMT d</p> <p>[%s] Created consumer consid=%PRINTF_LLFMTd connid=%PRINTF_LLFMTd topic=\q%s\q</p> <p>Multicast Daemon Id=%s</p> <p>Statistics enabled on a %d second interval.</p> <p>Statistics disabled.</p> <p>Rotating log from %s to %s</p> <p>Memory allocation error, possible data loss.</p> <p>Unrecoverable PGM error rc=%d, reason=%s</p> <p>Could not parse configuration file \q%s\q</p> <p>Interface IP address: %s</p> <p>Tracing enabled.</p> <p>Tracing disabled.</p> <p>refused new connection with existing ID % PRINTF_LLFMT d</p> <p>[%s] Connection destroyed, connid=%PRINTF_LLFMTd</p> <p>Error sending to consid=%PRINTF_LLFMTd connid=%PRINTF_LLFMTd from channel \q%s\q: %s</p> <p>%s, status=%s</p> <p>Attached channel \q%s\q to consumer consid=%PRINTF_LLFMTd connid=%PRINTF_LLFMTd</p> <p>Error attaching channel \q%s\q to consumer consid=%PRINTF_LLFMTd connid=%PRINTF_LLFMTd</p>

Detaching channel \q%\q from consumer consid=%PRINTF_LLFMtd
connid=%PRINTF_LLFMtd
Destroying consumer consid=%PRINTF_LLFMtd connid=%PRINTF_LLFMtd
Channel configuration from server does not match existing channel \q%\q
Ignoring additional PGM receiver created on group \q%\q, dport=%d,
sport=%d, channel=%s
Created channel: \q%\q
Error: %s is not a valid multicast-capable IP address. Use the -ifc command line
parameter to specify a valid interface.

Category	Out of memory
Description	The server failed to allocate memory as it was attempting to perform an operation.
Resolution	Check how much memory the server process is using according to the operating system. Compare this with how much memory and swap space the host actually has. If there are sufficient memory and swap space, check the operating system limits on the server process to determine if this is the cause. If the limits are set to their maximum and this error occurs, reduce the load on this server by moving some topics and queues to another server.
Errors	<p>%s trying to recreate persistent message.</p> <p>Error during routed queue configuration, can not create routed queue consumer</p> <p>Could not initialize monitor</p> <p>Error: out of memory processing admin request</p> <p>Error during route configuration, can not create routed queue consumer, err=%s</p> <p>Configuration error - duplicate group: file=%s, line=%d: ignoring line</p> <p>Unable to create admin group: out of memory during initialization</p> <p>Error: unable to create alias for %s '%s': no memory</p> <p>Error: unable to create alias: out of memory</p> <p>Unable to create import event for %s '%s' on transport '%s'</p> <p>Unable to create internal connection, error=(%d) %s</p> <p>Unable to create internal connection: out of memory during initialization</p> <p>Error: unable to create %s '%s': no memory</p>

Error: unable to create route while parsing file=%s, line=%d.

Unable to create SmartSockets subscriber on transport '%s', %s '%s': out of memory

Unable to create temporary destination, out of memory

Failed to create reserve memory. Exiting.

Failed writing message to '%s': No memory for operation.

Unable to process message imported on transport '%s'.

Fault Tolerant configuration, no memory!

Fault Tolerant error, no memory.

LDAP initialization failed.

No memory.

No memory authenticating user '%s'

No memory authenticating via LDAP

Out of memory while building admin response message

Out of memory while building JNDI response message

Out of memory creating global import event on transport '%s'

Out of memory creating import event for %s '%s' on transport '%s'

Out of memory creating SS transport %s

No memory creating stalled flows in destination

Out of memory during initialization

Could not set replyto destination exporting SS message.

Could not set destination exporting SS message.

Could not get destination exporting SS message.

Failed to initialize SS message fields exporting SS message.

Out of memory exporting SS message.

Out of memory: unable to process SS message type on export

No memory for creating connection.

No memory generating dynamic route durable.

Out of memory importing SS message. error=%s.

No memory in IO thread to create pool.

Out of memory while parsing bridges file

Out of memory while parsing factories file

Out of memory while parsing routes file

No memory performing routing operation.

Out of memory processing %s on %s '%s'

Out of memory processing administrative request

Out of memory processing message tracing

No memory processing purge record.

No memory while processing route interest

Out of memory processing transports

Out of memory processing transports configuration

Out of memory reading configuration.

Out of memory restoring routed consumer

Out of memory sending monitor message.

No memory sending topic routing information.

%s trying to add message to %s queue.

No memory trying to add message to system.

No memory trying to cleanup route.

No memory to create ack record.

No memory to create client connection

No memory to create configured durable '%s%s%s'.

No memory to create configured durables

No memory to create confirm record.

No memory to create connection.

No memory to create consumer.

No memory trying to create destination.

No memory to create destination for consumer or browser.

No memory trying to create global topic destination.

No memory to create message from store.

No memory trying to create message producer.

No memory to create producer.
 No memory trying to create queue browser.
 No memory trying to create response message.
 No memory to create routed consumer
 No memory to create routed queue consumers
 No memory trying to create routed queue destination.
 No memory trying to create routed tmp queue destination.
 No memory to create session.
 No memory trying to create tmp destination for consumer.
 No memory trying to create transaction.
 No memory to create valid messages record.
 No memory restoring valid sequence number info.
 No memory to create zone.
 No memory trying to export message to RV.
 No memory trying to export message to SS.
 No memory trying to import message from RV%s.
 No memory trying to import message from RVCN.
 No memory trying to import message from SS. error=%s.
 No memory trying to initialize connection.
 No memory trying to initialize route connection.
 No memory trying to parse configured durable.
 No memory trying to process data message.
 No memory trying to process queue message.
 No memory to process route interest
 No memory trying to process system request.
 No memory trying to process topic consumer.
 No memory trying to process topic message.
 No memory trying to process xa end. connID=% PRINTF_LLfmt d sessID=%
 PRINTF_LLfmt d %s
 No memory trying to read message from store.

Route down while trying to recover routed consumer.

No memory trying to recover routed consumer.

No memory trying to recover one msg for routed consumer.

No memory trying to recover route stall.

No memory trying to recover route stall, will try again.

No memory to restore messages.

No memory to restore prepared transactions.

No memory trying to retrieve for queue browser.

No memory trying to send recover/rollback response.

out of memory trying to send topic interest to routes

No memory to set clientID for connection.

No memory trying to setup queue route configuration

No memory trying to setup route configuration

No memory trying to setup topic route configuration

Route recovery of destination %s on route from %s will fail: No memory

Route recovery of destination %s on route from %s will fail: No memory to create timer

Route recovery of destination %s on route from %s will fail: %s

Failed to initialize OpenSSL environment: out of memory

Out of memory queuing imported message for processing.

Out of memory gathering consumers for incoming message.

Out of memory scheduling message delete.

Out of memory preparing to write message.

Out of memory assembling list of message to store.

Out of memory processing route consumer.

Out of memory preparing message for writing.

Out of memory creating connection thread list.

Out of memory creating RV gateway thread list.

Out of memory creating SmartSockets gateway thread list. error=%s.

Out of memory delaying bridged flow control response.

Out of memory preparing to delay flow control response.

Out of memory delaying one flow control response.

Out of memory delaying set of flow control responses.

Out of memory trying to clear message hold.

Out of memory trying to delete held message.

Unable to update the valid messages record. Error code: %d - %s.

No memory scheduling message delete completion, Error code: %d.

No memory to build msg properties.

No memory to create prop.

No memory to set prop.

No memory getting the list of delivered messages. The JMSXDeliveryCount property of some messages may no longer be accurate.

No memory getting the list of delivered messages from session %
PRINTF_LL_FMT d. The JMSXDeliveryCount property of messages that were sent
to this session may no longer be accurate.

No memory getting the list of delivered messages during rollback of transaction
with xid: %s. The JMSXDeliveryCount property of messages that were rolled-back
may no longer be accurate.

Out of memory discarding message.

Out of memory advancing queue pending.

Out of memory adding message to pending list.

Out of memory returning message to pending list.

Out of memory trying to re-queue after xa rollback.

Out of memory finalizing restored queue: %s.

Out of memory restoring queue flush state.

Out of memory detaching message during queue purge.

Out of memory removing message from queue.

Out of memory retrieving message by correlation id.

Out of memory scheduling cleanup of transaction ack: %s.

Out of memory setting message all acked: %s.

Out of memory cleaning up transaction: %s.

Out of memory updating sent state on ack.

Out of memory updating in-doubt state on ack.
Out of memory removing message from system.
Out of memory associating ack with data.
Out of memory associating ack with transaction.
Out of memory resetting mstore discard scan: %s.
Out of memory recording modified topic.
Out of memory re-queuing sent messages.
No memory trying to resend delivered messages following an xa end NOTA.
connID=% PRINTF_LLFMT d sessID=% PRINTF_LLFMT d %s
No memory to create consumer on %s [%s]
Failed to set delivery count in%smessage: status=%s
Failure to create per-mstore delayed delivery state: %s.

Category	Protocol error, incorrect XID in XA request
Description	The tibemsd received an XA End instruction from the third party Transaction Manager which referred to a different transaction from the one currently in use by the session.
Resolution	Report this to the your Transaction Manager vendor.
Errors	Incorrect xid in xa end (0x%x) request. connID=% PRINTF_LLFMT d sessID=% PRINTF_LLFMT d %s

Category	Protocol error, transaction in incorrect state
Description	A client application's attempt to start an XA transaction failed because the transaction already exists and is not in the correct state.
Resolution	This error is most likely caused by an external transaction manager that allowed two separate client applications to use the same XA transaction identifier (XID). Consult the manual for the transaction manager or report this to the transaction manager vendor.
Errors	Cannot process xa start for a session when another transaction is already active on that session. connID=% PRINTF_LLFMT d sessID=% PRINTF_LLFMT d %s

Cannot process xa start with TMNOFLAGS when the transaction is already active. connID=% PRINTF_LL_FMT d sessID=% PRINTF_LL_FMT d %s

All clients participating in the same global transaction must use the same protocol, connID=% PRINTF_LL_FMT d

Invalid xa start (resume) request: the session was not previously suspended. connID=% PRINTF_LL_FMT d sessID=% PRINTF_LL_FMT d %s

Error processing xa start - transaction marked ROLLBACKONLY. connID=% PRINTF_LL_FMT d sessID=% PRINTF_LL_FMT d %s

Error processing xa start request, %s. connID=% PRINTF_LL_FMT d sessID=% PRINTF_LL_FMT d %s

Invalid xa end (suspend) request: session already suspended or not started. connID=% PRINTF_LL_FMT d sessID=% PRINTF_LL_FMT d %s

Invalid xa end request: the session was neither associated with a transaction nor suspended. connID=% PRINTF_LL_FMT d sessID=% PRINTF_LL_FMT d %s

Error processing xa prepare - transaction marked ROLLBACKONLY, %s. connID=% PRINTF_LL_FMT d %s

Category	Protocol message format error
Description	tibemspd received a message with either missing or incomplete data.
Resolution	Send details of the error and the situation in which it occurred to TIBCO Support.
Errors	<p>Unable to confirm session, invalid request.</p> <p>Unable to create consumer, invalid destination.</p> <p>Unable to init session, invalid request.</p> <p>Unable to process msg for export. error=%s.</p> <p>Unable to recover consumer, invalid request.</p> <p>Unable to recover consumer, invalid session.</p> <p>Unable to recover one msg for consumer, invalid request.</p> <p>Unable to recover one msg for consumer, invalid sequence number.</p> <p>Unable to recover one msg for consumer, invalid session.</p> <p>Unable to serve the flow stall recover request from server %s, invalid request.</p> <p>Unable to start consumer, invalid consumer</p> <p>Unable to server the flow stall recover request from server %s, invalid consumer.</p>

Unable to unsubscribe consumer, invalid client request.

%s: %s failed: illegal to use %s [%s] in standby mode.

Invalid flag in xa end request. connID=% PRINTF_LLFMT d sessID=% PRINTF_LLFMT d %s

Invalid flag in xa start request. connID=% PRINTF_LLFMT d sessID=% PRINTF_LLFMT d %s

Invalid request to delete temporary destination: %s. connID=% PRINTF_LLFMT d

Invalid request to delete temporary destination: not owner connection.

Invalid xid in commit request.

Invalid xid in commit transaction record.

Invalid xid trying to update(%d) transaction record.

Invalid xid in rollback request.

Invalid xid in rollback transaction record.

Invalid xid in xa commit request. connID=% PRINTF_LLFMT d

Invalid xid in xa end request. connID=% PRINTF_LLFMT d sessID=% PRINTF_LLFMT d

Invalid xid in xa prepare request. connID=% PRINTF_LLFMT d

Invalid xid in xa rollback request. connID=% PRINTF_LLFMT d

Invalid xid in xa start request. connID=% PRINTF_LLFMT d sessID=% PRINTF_LLFMT d

Malformed routed message

Problem decoding sequence data in confirm: %s.

Problem decoding sequence data in rollback.

Problem decoding sequence data in xa end. connID=% PRINTF_LLFMT d sessID=% PRINTF_LLFMT d %s

%s:%s queue browser failed: queue name is missing in request message

Received admin request with replyTo not set

Received JNDI request with replyTo not set.

Received unexpected message type %d

No destination in incoming data message.

Invalid %s message

Category	Protocol sequence error
Description	A non-embedded java client is attempting to connect to a tibemsd that is part of an embedded JMS environment.
Resolution	Reconfigure the client to connect to a fully licensed tibemsd.
Errors	Invalid client connect detected. No closure.

Category	Recovery errors
Description	An error occurred during the recovery process.
Resolution	If you are not able to fix the problem and need to restart the system, make a backup of the store files and restart the server with the '-forceStart' command line parameter. The server will then attempt to start regardless of errors (expect out-of-memory errors). In this mode, application messages and/or internal records causing problems (due to file corruption or other) will be deleted. Therefore, dataloss is likely to occur, so this command line parameter should be used with extreme caution and only after understanding the consequences. A copy of the store files can be sent to TIBCO Support for post-mortem analysis.
Errors	<p>The recovery process stopped while processing a '%s' record (id=% PRINTF_LL_FMT d), error: %d - %s. Check the section 'Error Recovery Policy' from chapter 'Running the EMS Server' in the User's Guide before attempting to restart the server</p> <p>The recovery process stopped while processing a '%s' record (id=% PRINTF_LL_FMT d) due to an out-of-memory condition. Ensure that the system can allocate sufficient memory to the EMS Server process before restarting it</p> <p>Unable to get the session's context handle for %s record: %d - %s</p> <p>Unable to get the list iterator for %s record</p> <p>Unable to get next element from list for %s record</p> <p>Unable to create %s object, no memory</p> <p>Error occurred while processing %s record id=% PRINTF_LL_FMT d (%s) - Unable to reconstruct message: %d - %s</p> <p>Unable to recreate zone '%s': %d - %s</p> <p>Unable to add zone '%s' to the system: %d - %s</p>

Zone '%s' is defined as type '%s' in configuration but also is defined as type '%s' in meta.db

Unable to recreate connection id=% PRINTF_LLFMT d: %d - %s

Discarding session id=% PRINTF_LLFMT d because the connection id=% PRINTF_LLFMT d was not recovered. Recovery continues

Unable to recreate session id=% PRINTF_LLFMT d with connection id=% PRINTF_LLFMT d: %d - %s

Unable to recreate consumer id=% PRINTF_LLFMT d with connection id=% PRINTF_LLFMT d and session id=% PRINTF_LLFMT d: invalid destination: %s

No memory to create destination for consumer id=% PRINTF_LLFMT d

Discarding consumer id=% PRINTF_LLFMT d on destination '%s' because connection id=% PRINTF_LLFMT d was not restored. Recovery continues

Discarding consumer id=% PRINTF_LLFMT d on destination '%s' and connection id=% PRINTF_LLFMT d because session id=% PRINTF_LLFMT d was not restored. Recovery continues

No memory to recreate consumer id=% PRINTF_LLFMT d

Failed to build import selectors for consumer id=% PRINTF_LLFMT d: %d - %s

Failed to read import selectors for routed consumer id=% PRINTF_LLFMT d: %d - %s

Discarding durable id=% PRINTF_LLFMT d (connection id=% PRINTF_LLFMT d) on destination '%s' because the durable name is not specified. Recovery continues

Unable to recreate producer id=% PRINTF_LLFMT d with connection id=% PRINTF_LLFMT d and session id=% PRINTF_LLFMT d: invalid destination: %s

No memory to create destination for producer id=% PRINTF_LLFMT d

Discarding producer id=% PRINTF_LLFMT d on destination '%s' because connection id=% PRINTF_LLFMT d was not restored. Recovery continues

Discarding producer id=% PRINTF_LLFMT d on destination '%s' with connection id=% PRINTF_LLFMT d because session id=% PRINTF_LLFMT d was not restored. Recovery continues

Unable to recreate purge record: invalid destination: %s

Unable to recreate purge record for destination %s: %d - %s

Error creating message for transaction record: %d - %s

Error creating message's store structure for transaction record: %d - %s

Unable to recover transaction record: transaction id missing: %d - %s

Unable to recover transaction id=% PRINTF_LLFMT d: %d - %s

Unable to recover ack record (txid=% PRINTF_LLFMT d, consid=% PRINTF_LLFMT d, seqid=% PRINTF_LLFMT d, location=%s): %d - %s

Unable to recover ack record, cannot create message: %d - %s

Unable to recover sequence numbers from valid record: %s

Unable to recover message, can not create lock: %d - %s

Unable to restore held message from store, (location=%s) no memory

Unable to restore message sequence=% PRINTF_LLFMT d: (location=%s) %d - %s

No memory to create destination for message

Inconsistency restoring routed message sequence=% PRINTF_LLFMT d

No memory to restore routed message sequence=% PRINTF_LLFMT d

Persisted message possibly corrupted: %s

Error creating message's store structure: %d - %s

Category	Rejected attempt to connect via SSL to TCP port
Description	A client application attempted to connect to the server's TCP port using the SSL protocol.
Resolution	Change the client application's URL from ssl to tcp or change the server's listen parameter from tcp to ssl. To activate a change of the server listen parameter requires a restart of the server.
Errors	Rejected attempt to connect via SSL to TCP port

Category	Rejected attempt to connect via TCP to SSL port
Description	A client application attempted to connect to the server's SSL port using the TCP protocol.
Resolution	Change the client application's URL from tcp to ssl or change the server's listen parameter from ssl to tcp. To activate a change of the server listen parameter requires a restart of the server.
Errors	Rejected attempt to connect via TCP to SSL port

Category	rejected connect from route: invalid cycle in route
Description	The multi-hop route support of the server does not support configuring a cycle. However, it detected a configuration that would create a cycle.
Resolution	Remove one of the routes that creates the cycle.
Errors	<p>[%s@%s]: rejected connect from route: invalid cycle in route: %s</p> <p>Illegal, route to '%s' creates a cycle. Terminate the connection</p> <p>Illegal, route to '%s' creates a cycle.</p>

Category	Rendezvous transport error
Description	tibemsd encountered a Rendezvous error.
Resolution	See Rendezvous documentation for details of what the error means and how to remedy it.
Errors	<p>Unable to create dispatcher for import event for %s '%s' on transport '%s', error is %s</p> <p>Unable to create inbox for import event for %s '%s' on transport '%s'</p> <p>Unable to create Rendezvous Certified transport '%s': %s</p> <p>Unable to create Rendezvous Certified transport '%s' because unable to create Rendezvous transport '%s'</p> <p>Unable to create Rendezvous transport '%s': %s</p> <p>Unable to create TIBCO Rendezvous Certified Listener for %s '%s' on transport '%s': %s</p> <p>Failed to confirm RVCN message: %d (%s)</p> <p>Failed to confirm RVCN message sequence % PRINTF_LLFMT u from cm sender '%s'. Error: %d (%s)</p> <p>Unable to store trackId % PRINTF_LLFMT d for RVCN message sequence % PRINTF_LLFMTu from cm sender '%s'. Error: %d (%s)</p> <p>Unable to retrieve trackId % PRINTF_LLFMT d. Error: %d (%s)</p> <p>A problem occurred while importing RVCN message sequence % PRINTF_LLFMT u from cm sender '%s'. Expecting a redelivery</p> <p>Unable to queue the request type: %d. Transport '%s', destination '%s', CM Sender '%s', CM Sequence % PRINTF_LLFMT u . Error: %d (%s)</p>

Unable to queue the request type: %d. Transport '%s', destination '%s'. Error: %d (%s)

Failed to disallow Rendezvous Certified Message listener '%s': %s

Unable to export topic message, error=%s.

Unable to pre-register certified listener '%s' on transport '%s': %s

Rendezvous send failed on transport '%s', error='%s'

Unable to restart the CM Listener for %s '%s' (RVCM Transport '%s'). Error code: %d '%s'

Unable to create the timer for the restart of the CM Listener for %s '%s' (RVCM Transport '%s'). Error code: %d '%s'

Unable to stop the CM Listener for %s '%s' (RVCM Transport '%s'). Error code: %d '%s'

Category	Restoring consumer failed
Description	Seen when tibemspd starts up and detects that the zone for a route as specified in routes.conf has been changed.
Resolution	Either delete the route or change its zone back and restart the tibemspd.
Errors	Restoring consumer failed: Conflicting zone for route to [%s]: The route was initially zone %s type %s, but now %s type %s. Zone change not allowed while there are durable subscribers. Please delete the route first and create new one.

Category	Running on reserve memory
Description	Warnings indicating that the tibemspd has run out of memory and is now using its reserve memory
Resolution	Either reduce the tibemspd's memory requirement by consuming messages or removing messages from its queues, or increase the amount of memory available to the tibemspd by shutting down other processes on the machine or increasing the machine's memory.
Errors	Running on reserve memory, ignoring new message. Running on reserve memory, no more send requests accepted. Pending msg count = % PRINTF_LLGMT d Pending msg count = % PRINTF_LLGMT d

Category	Runtime Error in Fault-Tolerant Setup
Description	In a fault-tolerant setup, error occurs at runtime.
Resolution	Check the status of the both server (primary, standby). In case of both active, the file store data may be corrupted already and we recommend shutting down both servers and investigate the situation.
Errors	<p>Fault-tolerance error: Dual-Active server detected at: '%s'</p> <p>The primary EMS server does not hold the lock on meta store</p> <p>The standby EMS server could not find the specified meta store.</p> <p>The primary EMS server name is %s while the standby EMS server name is %s. The names must be the same</p> <p>A backup EMS server (%s) is already connected to the primary EMS server</p> <p>Fault Tolerant error (%s), can't create connection to '%s'.</p>

Category	SmartSockets transport error
Description	tibemsd encountered a SmartSockets error.
Resolution	See SmartSockets documentation for details of what the error means and how to remedy it.
Errors	<p>Unable to create SmartSockets subscriber on transport '%s': failed to convert %s '%s', error=%s</p> <p>Unable to import SmartSockets message on transport %s: failed to convert subject '%s', error=%s</p> <p>Unable to import SmartSockets message on transport %s: failed to tokenize subject '%s'</p> <p>Unable to import SmartSockets message on transport %s: failed to convert reply subject '%s', error=%s</p> <p>Unable to import SmartSockets message on transport %s: no destination found '%s'</p> <p>Unable to export EMS message into SmartSockets on transport '%s'. Failed to convert subject '%s', error=%s.</p> <p>Unable to export EMS message into SmartSockets on transport '%s'. Failed to convert reply subject '%s', error=%s.</p>

Error translating EMS message body into SS message. Status=%s

Error translating EMS message headers into SS message. Status=%s

Error translating EMS message properties into SS message. Status=%s

Unable to confirm SS message. %s

Unable to connect to SmartSockets RTserver via transport: '%s': %d - %s

Unable to register GMD failure callback: '%s': %d - %s

Unable to create open callback on transport: '%s': %d - %s

Unable to create default callback on transport: '%s': %d - %s

Unable to create SS callback for %s '%s' on transport '%s' SS error: %s

Unable to create SS message type on export

Unable to create SmartSockets subscriber for %s '%s' on transport '%s', error: %s

Unable to create SmartSockets transport '%s': %d - %s

Failed to confirm SS message. error=%s.

Failed to create SmartSockets transport %s

Unable to handoff confirm SS message: %s.

Unable to import SS message. Error=%d, %s.

Unable to import SS message data fields. Error=%d, %s.

Unable to import SS message headers. Error=%d, %s.

Unable to import SS message, failed to create message destination.

Unable to import SS message, failed to create reply destination.

Unable to import SS message, error retrieving delivery mode.

Unable to import SS message, error setting imported property. error=%s.

Unable to import SS message, error setting message extensions property. error=%s.

Unable to import SS message, failed to create message wire. error=%s.

Unable to import SS message, error retrieving number of fields.

Unable to initialize SmartSockets transport '%s': error=%d: %s

Unable to set SmartSockets Dispatcher for transport: '%s': %d - %s

Unable to set SS message type on export

Unable to set Username/Password for SmartSockets transport '%s': %d - %s

Unable to import SmartSockets message on transport %s: failed to retrieve SS subject.

SS Subject CB destroy Failed: for '%s' on transport '%s' SS error: %s

SS Subject CB lookup Failed: for '%s' on transport '%s' SS error: %s

SmartSockets TipcMsgSetDeliveryMode failed, '%s'

SmartSockets TipcMsgSetLbMode failed, '%s'

SmartSockets TipcSrvConnFlush failed, '%s'

SmartSockets TipcSrvConnMsgSend failed, '%s'

SS Unsubscribe failed: for '%s' on transport '%s' SS error: %s

GMD delivery failed on transport '%s', SS message seq=%d, reason='%s' for process '%s'

Unable to process undelivered SS GMD message, can not register EMS message, error='%s', tport='%s', GMD seq=%d

Unable to process undelivered SS GMD message, can not add to undelivered EMS queue, error='%s', tport='%s', GMD seq=%d

Unable to process undelivered SS GMD message, failed to build EMS message, error='%s', tport='%s', GMD seq=%d

Unable to convert undelivered SS GMD message into EMS message, error='%s', tport='%s', GMD seq=%d

Category	SSL initialization failed
Description	The server failed attempting to initialize the OpenSSL library.
Resolution	Examine the OpenSSL error and the EMS User's Guide chapter describing the use of SSL.
Errors	Failed to process ft ssl password Failed to process ssl password Ignoring SSL listen port %s Failed to initialize SSL: can not load certificates and/or private key and/or CRL file(s) and/or ciphers. Failed to initialize OpenSSL environment: error=%d, message=%s. Failed to initialize SSL. Error=%s Failed to initialize SSL: unable to obtain password

Failed to initialize SSL: server certificate not specified.

Failed to initialize SSL: server private key not specified.

Category	Store file format mismatch
Description	The store files specified were created from a different version of EMS that is not supported by this version.
Resolution	Revert to use the version of EMS that created the store file or locate the store file conversion tool and use it to convert the store file to this version.
Errors	Unsupported store format: %s (%d)

Category	System call error, should be errno-driven
Description	A low-level system function has failed.
Resolution	Report the error to your system administrator and ask them to remedy the problem.
Errors	<p>Accept() failed: too many open files. Please check per-process and system-wide limits on the number of open files.</p> <p>Accept() failed: %d (%s)</p> <p>Select() failed: %d (%s)</p> <p>Epoll_wait() failed: %d (%s)</p> <p>Epoll_ctl() %s on fd %d failed: %d (%s)</p> <p>ioctl() on /dev/poll failed: %d (%s)</p> <p>write() %s update /dev/poll on fd %d failed: %d (%s)</p> <p>Cannot retrieve user name of the current process.</p> <p>Client connection not created, %s.</p> <p>Could not obtain hostname</p> <p>Could not resolve hostname '%s'. Possibly default hostname is not configured properly while multiple network interfaces are present.</p> <p>Unable to listen for connections: %d (%s).</p> <p>Unable to open socket for listening: %d (%s).</p>

Closing SSL connection from %s due to timeout, exceeded handshake_timeout of %d.

Could not %s sequential file optimization: %d.

Category	Transaction action while previous action is incomplete.
Description	State-modifying action is requested on a transaction for which another such action is being processed.
Resolution	Send details of the error and the situation in which it occurred to TIBCO Support.
Errors	Cannot request second state change for transaction while the first request is in progress (%d, %d) %s. Unexpected request to roll xa txn forward with previous operation (%d) incomplete: %s. Unexpected request to roll xa txn back with previous operation (%d) incomplete: %s. Unexpected request to prepare xa txn with previous operation (%d) incomplete: %s. Unexpected request to commit xa txn with previous operation (%d) incomplete: %s. Unexpected request to commit session with previous operation (%d) incomplete.

Category	Transaction timeout.
Description	Transaction not completed before timeout. Offending transaction is discarded.
Resolution	Most likely, transaction manager error prevented it from advancing this transaction in a timely manner. Verify correct operation of the transaction manner.
Errors	Rollback due to timeout on unprepared transaction. connID=% PRINTF_LLFMT d %s

Category	Unnecessary or duplicate message
Description	tibemsd received a message with either missing or incomplete data.
Resolution	Send details of the error and the situation in which it occurred to TIBCO Support.

Errors	Error processing xa start request, %s. connID=% PRINTF_LL_FMT d sessID=% PRINTF_LL_FMT d Error trying to enter standby for '%s', %s.
Category	Unrecognized option
Description	The server's command line contains an unrecognized option.
Resolution	Run the server with the -help option and compare it with the command line containing the unrecognized option.
Errors	Unrecognized option: '%s'.

Index

Symbols

.NET

assembly version [329](#)

programmer's checklist [328](#)

\$sys.redelivery.delay [70](#)

A

acknowledgement [39](#)

acl.conf file [239](#)

add member command [128](#)

addprop factory command [128](#)

addprop queue command [128](#)

addprop route command [129](#)

addprop topic command [129](#)

admin

connect [130](#)

password [117](#)

user [126](#)

admin user [117](#)

administrator

assign password [126](#)

anonymous

user and security [117](#)

architecture

multicast [388](#)

authorization parameter [199](#)

AUTO_ACKNOWLEDGE mode [39](#), [337](#)

autocommit command [129](#)

B

bandwidth

managing multicast [384](#)

bridges [82](#)

bridges.conf file [240](#)

C

C

programmer's checklist [322](#)

c#

assembly version [329](#)

programmer's checklist [328](#)

certificates

file names [469](#)

changes from the previous release [xxvi](#)

channel property [59](#)

channels

detailed statistics [461](#)

channels.conf file [241](#)

cipher suites

.NET clients [481](#)

Java clients [478](#)

client tracing [222](#)

CLIENT_ACKNOWLEDGE mode [39](#), [40](#)

client_heartbeat_server parameter [211](#)

client_timeout_server_connection parameter [213](#)

client_trace parameter [222](#)

clock_sync_interval parameter [211](#)

cm_name parameter [406](#)

command line options

multicast [376](#)

commit command [129](#)

compact command [130](#)

compiling samples [95](#)

compliant_queue_ack parameter [199](#)

compression, message [38](#)

Configuration [108](#)

- configuring
 - external directory for authentication 279
 - LDAP 279
- connect
 - admin 130
- connect command 130
- connection factories 332
 - parameters 245
- connections
 - network I/O 122
- connectivity
 - multicast 382
- console_trace parameter 221
- consumers
 - detailed statistics 461
- conventions
 - naming 127
- conversion, data type 45
- cores
 - allocation 121
- create bridge command 131
- create durable command 131
- create factory command 131
- create group command 131
- create jndiname command 131
- create queue command 132
- create route command 132
- create rvcmlistener command 132
- create topic command 133
- create user command 133
- customer support xxxiii

D

- daemon parameter 406
- data type conversion 45
- database store files
 - Schema Export Tool 312
- database stores 304
 - configuring 305
 - in stores.conf 307
 - in tibemsd.conf 306
- dbstore_classpath parameter 306
- dbstore_driver_dialect parameter 307
- dbstore_driver_name parameter 307
- deadlock
 - flow control 89
- default_ttl parameter 406
- definitions of properties 58
- defrag destinations 254
- delete all command 133
- delete bridge command 133
- delete connection command 134
- delete durable command 134
- delete factory command 134
- delete group command 134
- delete jndiname command 134
- delete message command 134
- delete queue command 135
- delete route command 135
- delete rvcmlistener command 135
- delete subscriber 134
- delete topic command 135
- delete user command 135
- deployment considerations
 - multicast 382
- destination
 - bridges 82
- destination bridges
 - flow control 89
- destination defrag feature
 - file_destination_defrag 254
- destination properties 58
- destination_backlog_swapout parameter 207
- detailed statistics 461
- detailed_statistics parameter 223
- disabled security 116
- disconnect command 136
- disconnect_non_acking_consumers parameter 199
- distributed transactions 14
- DTC 14
- dual-state failover 493
- DUPS_OK_ACKNOWLEDGE mode 39, 41
- durable subscriber 5, 139, 139
 - extensible security 298
- durables.conf file 244
- dynamic destinations 54
 - creating 340

dynamically creating destinations
wildcards 79

E

echo command 136
EMS server
 starting 108
 stopping 111
emsntsrc 112
error recovery policy 115
exception listener 338
exclusive property 59
exit command 136
expiration property 60
EXPLICIT_CLIENT_ACKNOWLEDGE mode 40, 40, 41, 41
EXPLICIT_CLIENT_DUPS_OK_ACKNOWLEDGE mode 41, 41
explicit_config_only parameter 406
export
 topic property 61
export property 61
export_headers parameter 407
export_properties parameter 407
extensible security
 parameters 235

F

factories.conf file 245
failover
 dual state 493
 non-shared state 486
 shared state 495
 shared state overview 486
 unshared state 492
fault tolerance 13, 485
fault-tolerant switchover 28, 505, 519
fault-tolerant URL 246

file names
 certificates and keys 469
file-based stores 30
files, sample 94
FIPS
 fips140-2 parameter 228
FIPS 140-2 483
flow control 87
 enforcing 87, 87
 threads and deadlock 89
 with destination bridges 89
 with multicast 88
 with routes 88
flow_control parameter 200
flowControl property 61, 87
ft_activation parameter 214
ft_active parameter 213
ft_heartbeat parameter 213
ft_reconnect_timeout parameter 214
ft_ssl_auth_only parameter 214
ft_ssl_ciphers parameter 217
ft_ssl_expected_hostname parameter 216
ft_ssl_identity parameter 214
ft_ssl_issuer parameter 215
ft_ssl_password parameter 215
ft_ssl_private_key parameter 215
ft_ssl_rand_egd parameter 216
ft_ssl_trusted parameter 215
ft_ssl_verify_host parameter 216
ft_ssl_verify_hostname parameter 216

G

global property 62
grant admin command 138
grant queue command 136
grant topic command 137
group 279
groups.conf file 249

H

handshake_timeout parameter 207
help command 138

I

ibrvcm.conf file 257
import
 queue property 62
 topic property 62
import property 62
info command 138
inheritance 80
 property 80
inheritance of property 54
intervals
 mstore 33

J

jaas_classpath parameter 235
jaas_config_file parameter 235
jaas_login_timeout parameter 235
jaas.conf file 250
jaci_class parameter 236
jaci_classpath parameter 236
jaci_timeout parameter 236
Java
 programmer's checklist 321
JMS specification
 1.1 319
 2.0 318
JMS_TIBCO_SS_CORRELATION_ID property 440
JMS_TIBCO_SS_DELIVERY_MODE property 440
JMS_TIBCO_SS_EXPIRATION property 440
JMS_TIBCO_SS_LB_MODE property 440
JMS_TIBCO_SS_MESSAGE_ID property 440
JMS_TIBCO_SS_PRIORITY property 440
JMS_TIBCO_SS_SENDER property 439
JMS_TIBCO_SS_SENDER_TIMESTAMP property 440

JMS_TIBCO_SS_SEQ_NUM property 440
JMS_TIBCO_SS_TYPE_NUM property 440
JMS_TIBCO_SS_USER_PROP property 440
jre_library parameter 237
jre_option parameter 237
JSON configuration 108
JVM
 parameters 237

K

key
 file names 469

L

LDAP 279
ldap_all_groups_filter parameter 233
ldap_all_users_filter parameter 232
ldap_cache_enabled parameter 229
ldap_cache_ttl parameter 229
ldap_conn_type parameter 229
ldap_credential parameter 229
ldap_dynamic_group_attribute parameter 234
ldap_dynamic_group_class parameter 234
ldap_dynamic_member_url_attribute parameter 235
ldap_group_base_dn parameter 232
ldap_group_filter parameter 233
ldap_group_scope parameter 232
ldap_principal parameter 229
ldap_static_group_attribute parameter 233
ldap_static_group_class parameter 233
ldap_static_group_member_filter parameter 234
ldap_static_member_attribute parameter 234
ldap_tls_cacert_dir parameter 230
ldap_tls_cacert_file parameter 230
ldap_tls_cert_file parameter 231
ldap_tls_cipher_suite parameter 230
ldap_tls_key_file parameter 231
ldap_tls_rand_file parameter 230
ldap_url parameter 228

- ldap_user_attribute parameter [231](#)
- ldap_user_base_dn parameter [231](#)
- ldap_user_class parameter [231](#)
- ldap_user_filter parameter [232](#)
- ldap_user_scope parameter [232](#)
- ledger_file parameter [406](#)
- length limitations
 - naming conventions [127](#)
- library files [323](#)
- linking [323](#)
- load balancing [4](#), [60](#), [82](#), [249](#), [360](#), [429](#)
- load balancing URL [246](#)
- log_trace parameter [220](#)
- logfile parameter [220](#)
- logfile_max_size parameter [221](#), [221](#)

M

- MapMessage [22](#), [352](#)
 - definition [22](#), [352](#)
- max_client_msg_size parameter [207](#)
- max_connections parameter [207](#)
- max_msg_field_print_size parameter [201](#)
- max_msg_memory parameter [208](#)
- max_msg_print_size parameter [201](#)
- max_stat_memory parameter [224](#)
- maxbytes property [63](#)
- maxmsgs property [64](#)
- maxRedelivery property [64](#)
- message
 - acknowledgement [39](#)
 - compression [38](#)
 - maximum size [23](#)
 - selectors [42](#)
 - tracing [452](#)
- message listener [49](#)
- message pool [208](#)
- messaging model
 - multicast [6](#)
 - point-to-point [3](#)
 - publish and subscribe [4](#)
- MS DTC [14](#)
- msg_pool_block_size parameter [208](#)

- msg_swapping parameter [209](#)
- mstore [31](#)
 - intervals [33](#)
- multicast
 - architecture [388](#)
 - backwards compatibility [373](#)
 - command line options [376](#)
 - connectivity [382](#)
 - daemon [375](#)
 - daemon errors [399](#)
 - deployment considerations [382](#)
 - determining network capacity [393](#)
 - example [104](#)
 - example deployment [388](#)
 - flow control [88](#)
 - managing bandwidth [384](#)
 - messaging model [6](#)
 - restricting traffic [384](#)
 - server errors [400](#)
 - wildcards [78](#)
- multicast daemon
 - errors [399](#)
- multicast parameter [217](#)
- multicast troubleshooting [397](#)
 - connectivity [398](#)
 - data loss [398](#)
 - general tips [397](#)
- multicast_channels parameter [218](#)
- multicast_daemon_default parameter [218](#)
- multicast_statistics_interval parameter [218](#)
- multiple stores [30](#)

N

- name length limitations [127](#)
- naming conventions [127](#)
- network I/O connections
 - tuning [122](#)
- network parameter [405](#)
- NO_ACKNOWLEDGE mode [40](#), [40](#), [40](#)
- No-Acknowledgement Receipt Mode [40](#)
- non-shared state [486](#)
 - implementing [507](#)

npsend_check_mode parameter [202](#)

O

options

tibemsadmin [124](#)

overflowPolicy property [65](#)

P

password

admin [117](#)

setting [126, 142](#)

password parameter [202](#)

performance tuning [121](#)

permission

protection [273](#)

secure property and [71](#)

persistent messages [27](#)

in queues [27](#)

in topics [27](#)

synchronous file storage [28](#)

PGM [6](#)

point-to-point

example [98](#)

messaging model [3](#)

policy.1.0 [329](#)

prefetch property [68](#)

processor_ids parameter [203](#)

producers

detailed statistics [461](#)

programmer checklists [321](#)

C [322](#)

C# [328](#)

Java [321](#)

properties

channel [59](#)

definitions [58](#)

exclusive [59](#)

expiration [60](#)

export [61](#)

flowControl [61](#)

global [62](#)

import [62](#)

maxbytes [63](#)

maxmsgs [64](#)

maxRedelivery [64](#)

overflowPolicy [65](#)

prefetch [68](#)

queue [58](#)

redeliveryDelay [70](#)

secure [71](#)

sender_name [72](#)

sender_name_enforced [72](#)

store [73](#)

topic [58](#)

trace [74](#)

property [16, 16, 16, 16](#)

export [61](#)

import [62](#)

inheritance [54, 80](#)

protection permissions [273](#)

publish and subscribe

example [99](#)

messaging model [4](#)

purge all queues command [139](#)

purge all topics command [139](#)

purge durable command [139](#)

purge queue command [139](#)

purge topic command [139](#)

Q

queue import property [62](#)

queue limit policy [404](#)

queue properties [58](#)

queue property list [58](#)

queue_import_dm parameter [407](#)

- queues
 - delayed message redelivery 70
 - dynamic 54
 - routed 528
 - static 54
 - temporary 54
 - undelivered messages 21
- queues.conf file 250

R

- rate_interval parameter 223
- redeliveryDelay property 70
- remove member command 139
- removeprop factory command 140
- removeprop queue command 140
- removeprop route command 140
- removeprop topic command 140
- request_old parameter 406
- reserve memory 209
- reserve_memory parameter 209
- restricting multicast traffic 384
- resume route command 140
- revoke admin command 140
- revoke queue command 141
- revoke topic command 141
- rotatelog command 142
- round-robin queue (non-exclusive) 60
- routes
 - detailed statistics 461
 - flow control 88
 - zone 514
- routes.conf file 251
- rv_tport parameter 406
- RVC parameters 406

S

- sample files 94
- samples
 - compiling 95

- Schema Export Tool 312
- secure property 71
- secure property and permission 71
- security
 - and anonymous user 117
 - disabled 116
 - main configuration file 275
- selector_logical_operator_limit parameter 204
- selectors, message 42
- sender_name property 72
- sender_name_enforced property 72
- server
 - starting 108
 - stopping 111
- server parameter 204
- server_heartbeat_client parameter 212
- server_heartbeat_server parameter 212
- server_rate_interval parameter 222
- server_timeout_client_connection parameter 211
- server_timeout_server_connection parameter 212
- service parameter 405
- set password command 142
- set server command 143
- setprop factory command 148
- setprop queue command 148
- setprop route command 149
- setprop topic command 149
- shared state 495
 - process 488
- shared subscriptions 5
- show bridge command 149
- show bridges command 150
- show config command 152
- show connections command 156
- show db command 159
- show durable command 159
- show durables command 160
- show factories command 161
- show factory command 161
- show group command 162
- show groups command 162
- show jndiname command 162
- show jndinames command 162
- show members command 162
- show message command 163

- show messages command [163](#)
- show parents command [163](#)
- show queue command [164](#)
- show queues command [165](#)
- show route command [166](#)
- show routes command [167](#)
- show rvcmlisteners command [168](#)
- show rvcmtransportledger command [167](#)
- show server command [168](#)
- show stat command [168](#)
- show store command [169](#)
- show stores command [171](#)
- show subscriptions command [175](#)
- show topic command [171](#)
- show topics command [173](#)
- show transaction command [176](#)
- show transactions command [178](#)
- show transport command [179](#)
- show transports command [179](#)
- show user command [180](#)
- show users command [180](#)
- showacl admin command [180](#)
- showacl group command [180](#)
- showacl queue command [180](#)
- showacl topic command [181](#)
- showacl user command [181](#)
- shutdown command [181](#)
- socket_receive_buffer_size parameter [210](#)
- socket_send_buffer_size parameter [210](#)
- SSL
 - server parameters [224](#)
- ssl_auth_only parameter [228](#)
- ssl_cert_user_specname parameter [225](#)
- ssl_crl_path parameter [228](#)
- ssl_crl_update_interval parameter [228](#)
- ssl_dh_size parameter [224](#)
- ssl_password parameter [226](#)
- ssl_rand_egd parameter [227](#)
- ssl_require_client_cert parameter [225](#)
- ssl_server_ciphers parameter [224](#)
- ssl_server_identity parameter [226](#)
- ssl_server_issuer parameter [227](#)
- ssl_server_key parameter [226](#)
- ssl_server_trusted parameter [227](#)
- ssl_use_cert_username parameter [225](#)

- starting the EMS server [108](#)
- startup_abort_list parameter [205](#)
- static queues [54](#)
- static topics [54](#)
- statistics [460](#)
- statistics parameter [223](#)
- statistics_cleanup_interval parameter [224](#)
- stopping the EMS server [111](#)
- store files
 - configuring database stores [305](#)
 - configuring multiple stores [32](#)
 - database stores [304](#)
 - defaults [31](#)
 - destination defrag [254](#)
 - file-based stores [30](#)
 - mstore intervals [33](#)
 - show store command [169](#)
 - show stores command [171](#)
- store parameter [206](#)
- store property [73](#)
- stores
 - multiple stores [30](#)
- stores.conf
 - database store configuration [307](#)
- stores.conf file [253](#)
- subject collisions [411](#)
- subscriber [284, 284](#)
 - delete [134](#)
 - durable [139, 139](#)
- support, contacting [xxxiii](#)
- suspend route command [181](#)
- swapping
 - msg_swapping parameter [209](#)
- sync_ledger parameter [406](#)

T

- tcp [97, 130, 332](#)
- technical support [xxxiii](#)
- temp_destination_timeout parameter [408](#)
- temporary queues [54](#)
- temporary topics [54](#)

- threads [121](#)
 - flow control [89](#)
 - processor_ids parameter [203](#)
- throughput [121](#)
- TIBCO_HOME [xxx](#)
- tibemsadmin
 - options [124](#)
- tibemsd [323, 328](#)
- tibemsd.conf
 - database store configuration [306](#)
- tibemsd.conf file [187](#)
- tibemsmcd [375](#)
- tibemsMsg_GetStringProperty [353, 353](#)
- tibemsMsg_SetBooleanProperty [353, 353](#)
- tibrv_transports parameter [219](#)
- tibss_config_dir parameter [220](#)
- tibss_transports parameter [219](#)
- time command [182](#)
- topic export property [61](#)
- topic import property [62](#)
- topic properties [58](#)
- topic property list [58](#)
- topic_import_dm parameter [407](#)
- topics
 - dynamic [54](#)
 - routed [523](#)
 - static [54](#)
 - temporary [54](#)
- topics.conf file [257](#)
- trace property [74](#)
- trace_client_host parameter [222](#)
- tracing [452](#)
 - client [222](#)
 - trace options [448](#)
- track_correlation_ids parameter [217](#)
- track_message_id parameter [217](#)
- transaction commit command [182](#)
- transaction rollback command [182](#)
- transactions [14](#)
- transports.conf file [258](#)
- tuning
 - performance [121](#)
- type conversion [45](#)
- type parameter [405](#)

U

- undelivered message queue [21](#)
- UNIX system
 - using for user authentication [279](#)
- unshared state
 - configuring clients [507](#)
 - configuring servers [501](#)
 - process [492](#)
 - specifying URLs [508](#)
- updatecrl command [182](#)
- url formats [246](#)
- user [278](#)
 - admin [117](#)
 - externally authenticated [279](#)
- user admin [126](#)
- user_auth parameter [206](#)
- users.conf file [262](#)

W

- whoami command [183](#)
- wildcards [77](#)
 - in dynamically created destinations [79](#)
 - in queues [78](#)
 - in topics [78](#)

X

- xa_default_timeout parameter [206](#)

Z

- zones [514](#)